HumanActivityRecognition

This project is to build a model that predicts the human activities such as Walking, Walking_Upstairs, Walking_Downstairs, Sitting, Standing or Laying.

This dataset is collected from 30 persons(referred as subjects in this dataset), performing different activities with a smartphone to their waists. The data is recorded with the help of sensors (accelerometer and Gyroscope) in that smartphone. This experiment was video recorded to label the data manually.

How data was recorded

By using the sensors(Gyroscope and accelerometer) in a smartphone, they have captured '3-axial linear acceleration'(*tAcc-XYZ*) from accelerometer and '3-axial angular velocity' (*tGyro-XYZ*) from Gyroscope with several variations.

> prefix 't' in those metrics denotes time.
>
> suffix 'XYZ' represents 3-axial signals in X , Y, and Z directions.

Feature names

1. These sensor signals are preprocessed by applying noise filters and then sampled in fixed-width windows(sliding windows) of 2.56 seconds each with 50% overlap. ie., each window has 128 readings.
2. From Each window, a feature vector was obtianed by calculating variables from the time and frequency domain.

   > In our dataset, each datapoint represents a window with different readings

3. The accelertion signal was saperated into Body and Gravity acceleration signals(***tBodyAcc-XYZ*** and ***tGravityAcc-XYZ***) using some low pass filter with corner frequecy of 0.3Hz.
4. After that, the body linear acceleration and angular velocity were derived in time to obtian *jerk signals* (***tBodyAccJerk-XYZ*** and ***tBodyGyroJerk-XYZ***).
5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. This magnitudes are represented as features with names like *tBodyAccMag*, *tGravityAccMag*, *tBodyAccJerkMag*, *tBodyGyroMag* and *tBodyGyroJerkMag*.
6. Finally, We've got frequency domain signals from some of the available signals by applying a FFT (Fast Fourier Transform). These signals obtained were labeled with ***prefix 'f'*** just like original signals with ***prefix 't'***. These signals are labeled as ***fBodyAcc-XYZ***, ***fBodyGyroMag*** etc.,.
7. These are the signals that we got so far.

   - tBodyAcc-XYZ
   - tGravityAcc-XYZ
   - tBodyAccJerk-XYZ
   - tBodyGyro-XYZ
   - tBodyGyroJerk-XYZ
   - tBodyAccMag
   - tGravityAccMag
   - tBodyAccJerkMag
   - tBodyGyroMag
   - tBodyGyroJerkMag
   - fBodyAcc-XYZ
   - fBodyAccJerk-XYZ
   - fBodyGyro-XYZ
   - fBodyAccMag
   - fBodyAccJerkMag
   - fBodyGyroMag
   - fBodyGyroJerkMag
8. We can esitmate some set of variables from the above signals. ie., We will estimate the following properties on each and every signal that we recoreded so far.

   - ***mean()***: Mean value
   - ***std()***: Standard deviation
   - ***mad()***: Median absolute deviation
   - ***max()***: Largest value in array

- ***max()***: Largest value in array
- ***min()***: Smallest value in array
- ***sma()***: Signal magnitude area
- ***energy()***: Energy measure. Sum of the squares divided by the number of values.
- ***iqr()***: Interquartile range
- ***entropy()***: Signal entropy
- ***arCoeff()***: Autorregresion coefficients with Burg order equal to 4
- ***correlation()***: correlation coefficient between two signals
- ***maxInds()***: index of the frequency component with largest magnitude
- ***meanFreq()***: Weighted average of the frequency components to obtain a mean frequency
- ***skewness()***: skewness of the frequency domain signal
- ***kurtosis()***: kurtosis of the frequency domain signal
- ***bandsEnergy()***: Energy of a frequency interval within the 64 bins of the FFT of each window.
- ***angle()***: Angle between to vectors.

9. We can obtain some other vectors by taking the average of signals in a single window sample. These are used on the angle() variable' `

    - gravityMean
    - tBodyAccMean
    - tBodyAccJerkMean
    - tBodyGyroMean
    - tBodyGyroJerkMean

Y_Labels(Encoded)

- In the dataset, Y_labels are represented as numbers from 1 to 6 as their identifiers.
    - WALKING as **1**
    - WALKING_UPSTAIRS as **2**
    - WALKING_DOWNSTAIRS as **3**
    - SITTING as **4**
    - STANDING as **5**
    - LAYING as **6**

Train and test data were saperated

- The readings from ***70%*** of the volunteers were taken as ***trianing data*** and remaining ***30%*** subjects recordings were taken for ***test data***

Data

- All the data is present in 'UCI_HAR_dataset/' folder in present working directory.
    - Feature names are present in 'UCI_HAR_dataset/features.txt'
    - ***Train Data***
        - 'UCI_HAR_dataset/train/X_train.txt'
        - 'UCI_HAR_dataset/train/subject_train.txt'
        - 'UCI_HAR_dataset/train/y_train.txt'
    - ***Test Data***
        - 'UCI_HAR_dataset/test/X_test.txt'
        - 'UCI_HAR_dataset/test/subject_test.txt'
        - 'UCI_HAR_dataset/test/y_test.txt'

Data Size :

> 27 MB

Quick overview of the dataset :

- Accelerometer and Gyroscope readings are taken from 30 volunteers(referred as subjects) while performing the following 6 Activities.

    1. Walking
    2. WalkingUpstairs
    3. WalkingDownstairs
    4. Standing

4. Standing
5. Sitting
6. Lying.

- Readings are divided into a window of 2.56 seconds with 50% overlapping.
- Accelerometer readings are divided into gravity acceleration and body acceleration readings, which has x,y and z components each.
- Gyroscope readings are the measure of angular velocities which has x,y and z components.
- Jerk signals are calculated for BodyAcceleration readings.
- Fourier Transforms are made on the above time readings to obtain frequency readings.
- Now, on all the base signal readings., mean, max, mad, sma, arcoefficient, engerybands,entropy etc., are calculated for each window.
- We get a feature vector of 561 features and these features are given in the dataset.
- Each window of readings is a datapoint of 561 features.

Problem Framework

- 30 subjects(volunteers) data is randomly split to 70%(21) test and 30%(7) train data.
- Each datapoint corresponds one of the 6 Activities.

In [6]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.python.keras import backend as K
from keras.models import Sequential
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical
from keras.layers import Flatten
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout

# get the features from the file features.txt
features = list()
with open('UCI_HAR_Dataset/features.txt') as f:
    features = [line.split()[1] for line in f.readlines()]
print('No of Features: {}'.format(len(features)))
```

```
No of Features: 561
```

```
Using TensorFlow backend.
```

In [7]:

```
features_df = pd.DataFrame(features)
print('No of duplicate features: {}'.format(sum(features_df.duplicated())))
```

```
No of duplicate features: 84
```

In [8]:

```
print(list(features_df.columns))
```

```
[0]
```

Obtain the train data

In [9]:

```
# get the data from txt files to pandas dataframe
X_train = pd.read_csv('E:/BOOKS NEW/Cases datasets/7. Human Activity
Recognition/HAR/UCI_HAR_Dataset/train/X_train.txt', delim_whitespace=True, header=None)
```

```
Recognition/HAR/UCI_HAR_Dataset/train/X_train.txt', delim_whitespace=True, header=None)

# add subject column to the dataframe
X_train['subject'] = pd.read_csv('UCI_HAR_Dataset/train/subject_train.txt', header=None,
squeeze=True)

y_train = pd.read_csv('UCI_HAR_Dataset/train/y_train.txt', names=['Activity'], squeeze=True)
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})

# put all columns in a single dataframe
train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels
train.sample()
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	554	555	556
4932	0.273064	-0.005687	-0.09916	-0.991843	-0.942883	-0.858743	-0.992842	-0.94551	-0.859719	-0.937652	...	0.069997	0.0325	0.687609

1 rows × 564 columns

Obtain the train and test data

In [10]:

```
train = pd.read_csv('UCI_HAR_dataset/csv_files/train.csv')
test = pd.read_csv('UCI_HAR_dataset/csv_files/test.csv')
print(train.shape, test.shape)
```

```
(7352, 564) (2947, 564)
```

In [11]:

```
train.head(3)
```

Out[11]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 554	U
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.112754	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.053477	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.118559	

3 rows × 564 columns

In [12]:

```
# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train.ActivityName
```

In [13]:

```
# get X_test and y_test from test csv file
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test.ActivityName
```

In [14]:

```
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
```

```
X_train and y_train : ((7352, 561),(7352,))
X_test  and y_test  : ((2947, 561),(2947,))
```

Let's model with our data

Labels that are useful in plotting confusion matrix

In [15]:

```
labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UPSTAIRS']
```

Function to plot the confusion matrix

In [16]:

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
plt.rcParams["font.family"] = 'DejaVu Sans'

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Generic function to run any model specified

In [17]:

```
from datetime import datetime
def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=True, \
                 print_cm=True, cm_cmap=plt.cm.Greens):


    # to store results at various phases
    results = dict()

    # time at which model starts training
    train_start_time = datetime.now()
    print('training the model..')
    model.fit(X_train, y_train)
    print('Done \n \n')
    train_end_time = datetime.now()
    results['training_time'] =  train_end_time - train_start_time
    print('training_time(HH:MM:SS.ms) - {}\n\n'.format(results['training_time']))


    # predict test data
```

```
    print('Predicting test data')
    test_start_time = datetime.now()
    y_pred = model.predict(X_test)
    test_end_time = datetime.now()
    print('Done \n \n')
    results['testing_time'] = test_end_time - test_start_time
    print('testing time(HH:MM:SS:ms) - {}\n\n'.format(results['testing_time']))
    results['predicted'] = y_pred


    # calculate overall accuracty of the model
    accuracy = metrics.accuracy_score(y_true=y_test, y_pred=y_pred)
    # store accuracy in results
    results['accuracy'] = accuracy
    print('----------------------')
    print('|      Accuracy      |')
    print('----------------------')
    print('\n    {}\n\n'.format(accuracy))


    # confusion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    results['confusion_matrix'] = cm
    if print_cm:
        print('--------------------')
        print('| Confusion Matrix |')
        print('--------------------')
        print('\n {}'.format(cm))

    # plot confusin matrix
    plt.figure(figsize=(8,8))
    plt.grid(b=False)
    plot_confusion_matrix(cm, classes=class_labels, normalize=True, title='Normalized confusion
matrix', cmap = cm_cmap)
    plt.show()

    # get classification report
    print('-------------------------')
    print('| Classifiction Report |')
    print('-------------------------')
    classification_report = metrics.classification_report(y_test, y_pred)
    # store report in results
    results['classification_report'] = classification_report
    print(classification_report)

    # add the trained  model to the results
    results['model'] = model

    return results
```

Method to print the gridsearch Attributes

In [18]:

```
def print_grid_search_attributes(model):
    # Estimator that gave highest score among all the estimators formed in GridSearch
    print('--------------------------')
    print('|      Best Estimator     |')
    print('--------------------------')
    print('\n\t{}\n'.format(model.best_estimator_))


    # parameters that gave best results while performing grid search
    print('--------------------------')
    print('|     Best parameters     |')
    print('--------------------------')
    print('\tParameters of best estimator : \n\n\t{}\n'.format(model.best_params_))


    #  number of cross validation splits
    print('---------------------------------')
    print('|   No of CrossValidation sets   |')
```

```
    print('--------------------------------')
    print('\n\tTotal numbre of cross validation sets: {}\n'.format(model.n_splits_))


    # Average cross validated score of the best estimator, from the Grid Search
    print('--------------------------')
    print('|        Best Score       |')
    print('--------------------------')
    print('\n\tAverage Cross Validate scores of best estimator :
\n\n\t{}\n'.format(model.best_score_))
```

1. Logistic Regression with Grid Search

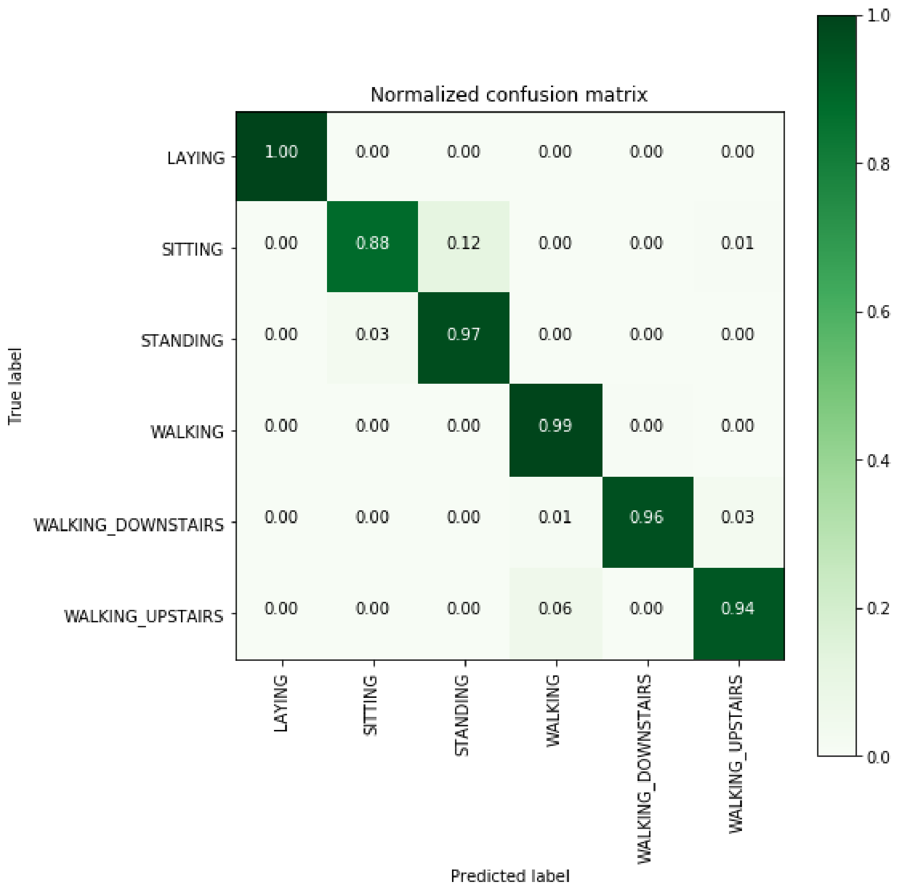
In [14]:

```
from sklearn import linear_model
from sklearn import metrics

from sklearn.model_selection import GridSearchCV

# start Grid search
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = linear_model.LogisticRegression()
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jobs=-1)
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test, y_test, class_labels=
labels)
```

```
training the model..
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:    8.8s finished
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
Done


training_time(HH:MM:SS.ms) - 0:00:10.278021


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.004989


---------------------
|      Accuracy      |
---------------------

    0.9582626399728538


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 431  57   0   0   3]
 [  0  15 517   0   0   0]
 [  0   0   0 493   2   1]
 [  0   0   0   4 403  13]
 [  0   0   0  27   1 443]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.88      0.92       491
          STANDING       0.90      0.97      0.93       532
           WALKING       0.94      0.99      0.97       496
WALKING_DOWNSTAIRS       0.99      0.96      0.98       420
  WALKING_UPSTAIRS       0.96      0.94      0.95       471

          accuracy                           0.96      2947
         macro avg       0.96      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947
```

2. Linear SVC with GridSearch

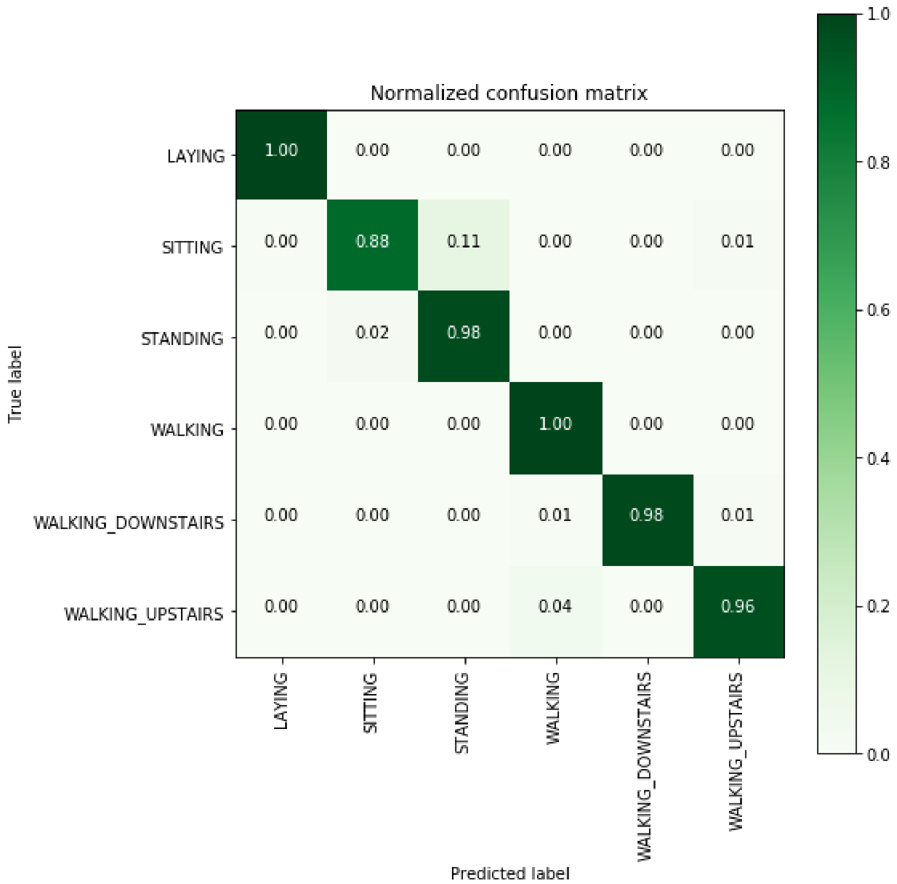
In [15]:

```
from sklearn.svm import LinearSVC

parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}
lr_svc = LinearSVC(tol=0.00005)
lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, verbose=1)
lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
training the model..
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:   25.7s finished
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:947: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
Done


training_time(HH:MM:SS.ms) - 0:00:29.332573


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.004958


---------------------
|      Accuracy      |
---------------------

    0.9670851713607058


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  2 432  53   0   0   4]
 [  0  12 519   1   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   3 412   5]
 [  0   0   0  17   0 454]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.88      0.92       491
          STANDING       0.91      0.98      0.94       532
           WALKING       0.96      1.00      0.98       496
```

```
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.96      0.97       471

          accuracy                           0.97      2947
         macro avg       0.97      0.97      0.97      2947
      weighted avg       0.97      0.97      0.97      2947
```

In [16]:

```python
print_grid_search_attributes(lr_svc_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 LinearSVC(C=0.5, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=5e-05,
          verbose=0)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator : 

 {'C': 0.5}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 5

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator : 

 0.9420643090682909
```

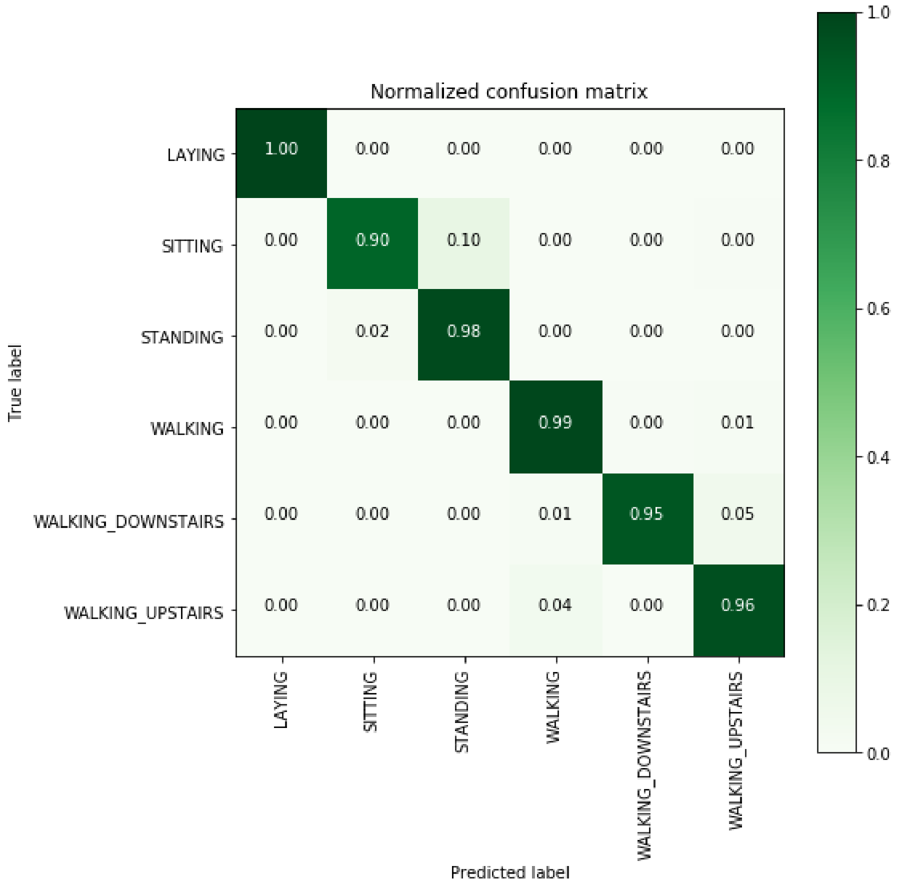
3. Kernel SVM with GridSearch

In [17]:

```python
from sklearn.svm import SVC
parameters = {'C':[2,8,16],\
              'gamma': [ 0.0078125, 0.125, 2]}
rbf_svm = SVC(kernel='rbf')
rbf_svm_grid = GridSearchCV(rbf_svm,param_grid=parameters, n_jobs=-1)
rbf_svm_grid_results = perform_model(rbf_svm_grid, X_train, y_train, X_test, y_test, class_labels=l
abels)
```

```
training the model..
Done 


training_time(HH:MM:SS.ms) - 0:03:54.956716


Predicting test data
Done 


testing time(HH:MM:SS:ms) - 0:00:02.075449


---------------------
|      Accuracy      |
---------------------
```

```
    0.9626739056667798


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 441  48   0   0   2]
 [  0  12 520   0   0   0]
 [  0   0   0 489   2   5]
 [  0   0   0   4 397  19]
 [  0   0   0  17   1 453]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.90      0.93       491
          STANDING       0.92      0.98      0.95       532
           WALKING       0.96      0.99      0.97       496
WALKING_DOWNSTAIRS       0.99      0.95      0.97       420
  WALKING_UPSTAIRS       0.95      0.96      0.95       471

          accuracy                           0.96      2947
         macro avg       0.96      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947
```

In [18]:

```python
print_grid_search_attributes(rbf_svm_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------
```

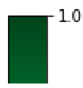
```
SVC(C=16, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
   decision_function_shape='ovr', degree=3, gamma=0.0078125, kernel='rbf',
   max_iter=-1, probability=False, random_state=None, shrinking=True,
   tol=0.001, verbose=False)

--------------------------
|      Best parameters     |
--------------------------
 Parameters of best estimator : 

 {'C': 16, 'gamma': 0.0078125}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 5

--------------------------
|        Best Score        |
--------------------------

 Average Cross Validate scores of best estimator : 

 0.9447834551903698
```

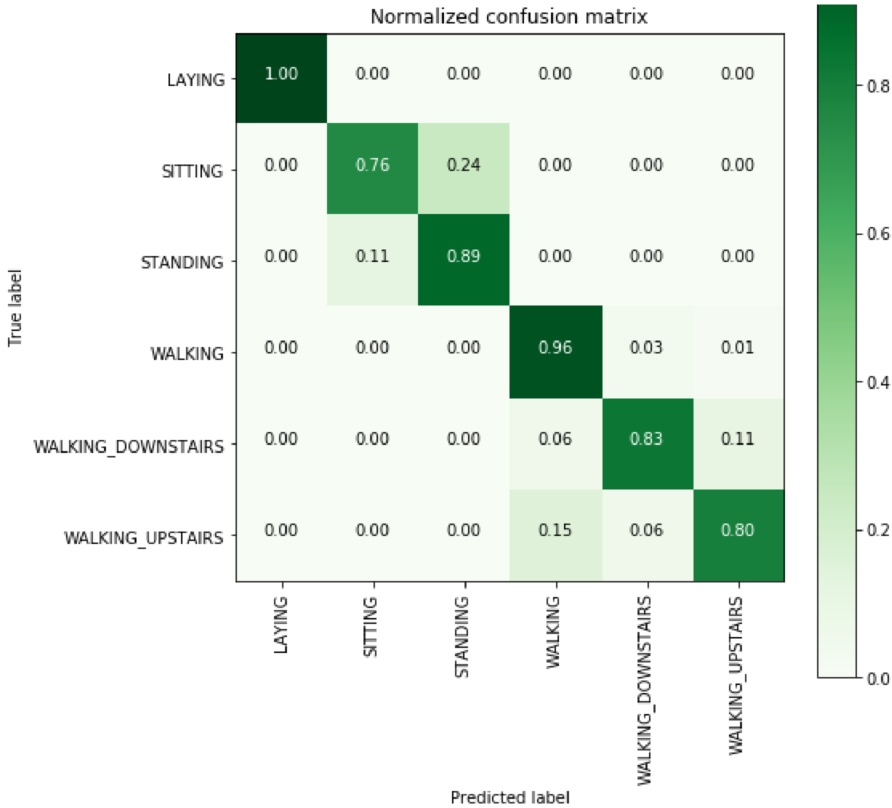
4. Decision Trees with GridSearchCV

In [19]:

```python
from sklearn.tree import DecisionTreeClassifier
parameters = {'max_depth':np.arange(3,10,2)}
dt = DecisionTreeClassifier()
dt_grid = GridSearchCV(dt,param_grid=parameters, n_jobs=-1)
dt_grid_results = perform_model(dt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
print_grid_search_attributes(dt_grid_results['model'])
```

```
training the model..
Done 
 

training_time(HH:MM:SS.ms) - 0:00:07.334382


Predicting test data
Done 
 

testing time(HH:MM:SS:ms) - 0:00:00.005984


---------------------
|      Accuracy      |
---------------------

    0.8747879199185612


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 372 119   0   0   0]
 [  0  61 471   0   0   0]
 [  0   0   0 474  16   6]
 [  0   0   0  24 349  47]
 [  0   0   0  70  26 375]]
```






```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.86      0.76      0.81       491
          STANDING       0.80      0.89      0.84       532
           WALKING       0.83      0.96      0.89       496
WALKING_DOWNSTAIRS       0.89      0.83      0.86       420
  WALKING_UPSTAIRS       0.88      0.80      0.83       471

          accuracy                           0.87      2947
         macro avg       0.88      0.87      0.87      2947
      weighted avg       0.88      0.87      0.87      2947

--------------------------
|      Best Estimator     |
--------------------------

 DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=9, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'max_depth': 9}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 5

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :
```

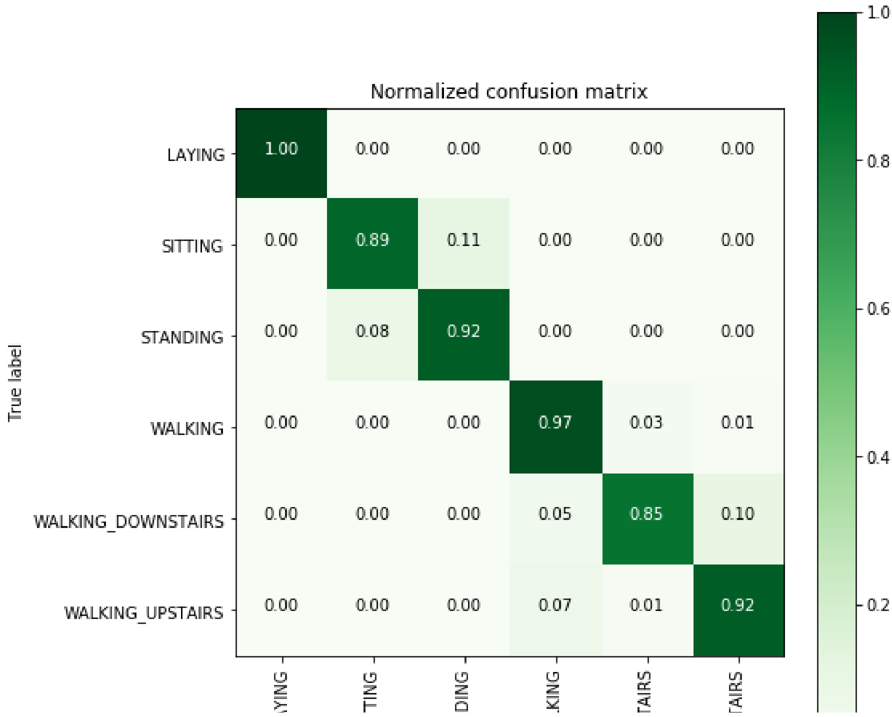
```
0.8512047429440844
```

5. Random Forest Classifier with GridSearch

In [20]:

```python
from sklearn.ensemble import RandomForestClassifier
params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3,15,2)}
rfc = RandomForestClassifier()
rfc_grid = GridSearchCV(rfc, param_grid=params, n_jobs=-1)
rfc_grid_results = perform_model(rfc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
print_grid_search_attributes(rfc_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:04:06.516162


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.077795


---------------------
|      Accuracy      |
---------------------

    0.9277231082456736


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 439  52   0   0   0]
 [  0  42 490   0   0   0]
 [  0   0   0 480  13   3]
 [  0   0   0  21 355  44]
 [  0   0   0  31   7 433]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.89      0.90       491
          STANDING       0.90      0.92      0.91       532
           WALKING       0.90      0.97      0.93       496
WALKING_DOWNSTAIRS       0.95      0.85      0.89       420
  WALKING_UPSTAIRS       0.90      0.92      0.91       471

          accuracy                           0.93      2947
         macro avg       0.93      0.92      0.93      2947
      weighted avg       0.93      0.93      0.93      2947

--------------------------
|      Best Estimator     |
--------------------------

 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=13, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=190,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'max_depth': 13, 'n_estimators': 190}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 5

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :

 0.9201615819679333
```

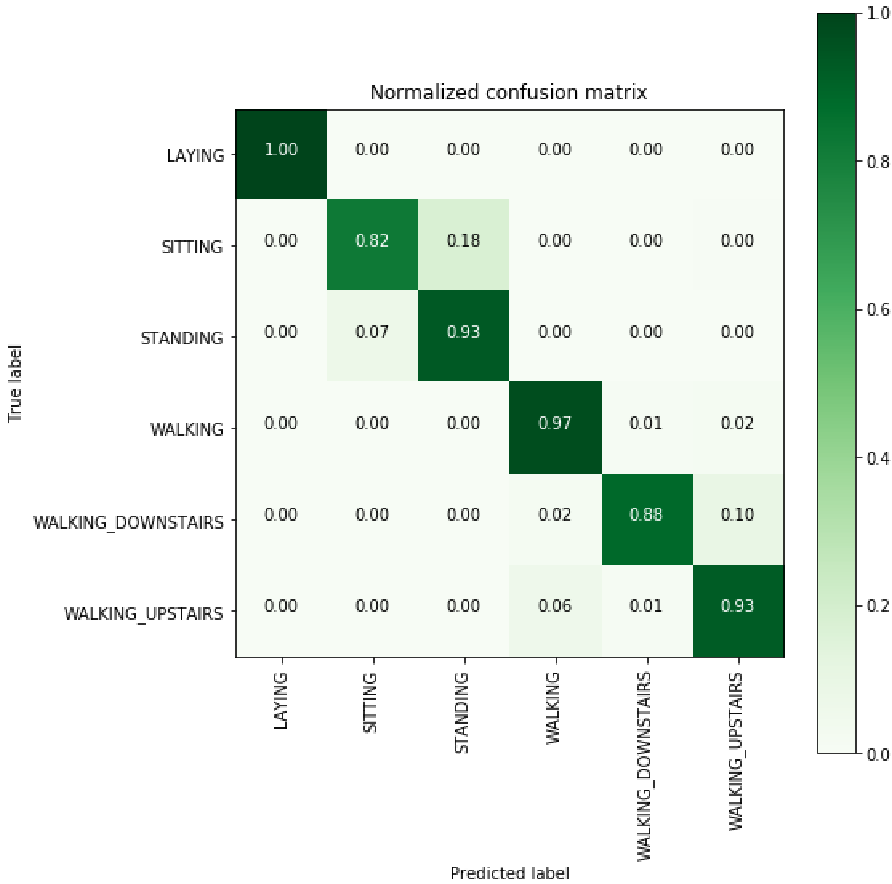
6. Gradient Boosted Decision Trees With GridSearch

In [21]:

```python
from sklearn.ensemble import GradientBoostingClassifier
param_grid = {'max_depth': np.arange(5,8,1), \
             'n_estimators':np.arange(130,170,10)}
gbdt = GradientBoostingClassifier()
gbdt_grid = GridSearchCV(gbdt, param_grid=param_grid, n_jobs=-1)
gbdt_grid_results = perform_model(gbdt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
print_grid_search_attributes(gbdt_grid_results['model'])
```

```
training the model..
Done
```

```
training_time(HH:MM:SS.ms) - 1:34:15.975005


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.053854


---------------------
|      Accuracy      |
---------------------

    0.9239904988123515


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 403  86   0   0   2]
 [  0  37 495   0   0   0]
 [  0   0   0 481   7   8]
 [  0   0   0   8 371  41]
 [  0   1   0  29   5 436]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.82      0.86       491
          STANDING       0.85      0.93      0.89       532
           WALKING       0.93      0.97      0.95       496
WALKING_DOWNSTAIRS       0.97      0.88      0.92       420
  WALKING_UPSTAIRS       0.90      0.93      0.91       471
```

```
             accuracy                           0.92      2947
            macro avg       0.93      0.92      0.92      2947
         weighted avg       0.93      0.92      0.92      2947

--------------------------
|      Best Estimator     |
--------------------------

 GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                            learning_rate=0.1, loss='deviance', max_depth=5,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=160,
                            n_iter_no_change=None, presort='deprecated',
                            random_state=None, subsample=1.0, tol=0.0001,
                            validation_fraction=0.1, verbose=0,
                            warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator : 

 {'max_depth': 5, 'n_estimators': 160}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 5

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator : 

 0.9155360091011252
```

7. Comparing all models

In [22]:

```
print('\n                     Accuracy     Error')
print('                     ----------   --------')
print('Logistic Regression : {:.04}%       {:.04}%'.format(log_reg_grid_results['accuracy'] * 100,\
                                                  100-(log_reg_grid_results['accuracy'] * 100)))

print('Linear SVC          : {:.04}%       {:.04}% '.format(lr_svc_grid_results['accuracy'] * 100,\
                                                        100-(lr_svc_grid_results['accuracy'] * 100)

print('rbf SVM classifier  : {:.04}%      {:.04}% '.format(rbf_svm_grid_results['accuracy'] * 100,\
                                                          100-(rbf_svm_grid_results['accuracy'] * 1
)))

print('DecisionTree        : {:.04}%      {:.04}% '.format(dt_grid_results['accuracy'] * 100,\
                                                        100-(dt_grid_results['accuracy'] * 100)))

print('Random Forest       : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy'] * 100,\
                                                           100-(rfc_grid_results['accuracy'] * 100)
print('GradientBoosting DT : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy'] * 100,\
                                                        100-(rfc_grid_results['accuracy'] * 100)))
```

```
                     Accuracy     Error
                     ----------   --------
Logistic Regression : 95.83%       4.174%
Linear SVC          : 96.71%       3.291%
rbf SVM classifier  : 96.27%      3.733%
DecisionTree        : 87.48%      12.52%
Random Forest       : 92.77%      7.228%
GradientBoosting DT : 92.77%      7.228%
```

> We can choose ***Logistic regression*** or ***Linear SVC*** or ***rbf SVM***.

8) Deep Learning Methods

8.1) LSTM

In [23]:

```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Utility function to print the confusion matrix
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

8.1.1 Data

In [26]:

```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

In [27]:

```
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]
```

In [26]:

```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None,encoding='utf-8')

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial_Signals/{signal}_{subset}.txt'
```

```
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(
            _read_csv(filename).values
            #_read_csv(filename).as_matrix()
            #coords = df.as_matrix(columns=['Latitude','Longitude'])
            #coords = df[["Latitude", "Longitude"]].values
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))
```

In [27]:

```
def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).values
```

In [28]:

```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test
```

In [29]:

```
# Importing tensorflow
np.random.seed(42)# The np.random.seed function provides an input for the pseudo-random number gen
erator in Python.
import tensorflow as tf
#tf.set_random_seed(42)
tf.random.set_seed(42)
```

In [30]:

```
# Configuring a session
#tf.ConfigProto

session_conf = tf.compat.v1.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [31]:

```
# Import Keras
#tf.Session
#tf.get_default_graph()
#from keras import backend as K
from tensorflow.python.keras import backend as K
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [32]:

```
# Importing libraries
from keras.models import Sequential
```

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout

# Initializing parameters
epochs = 30
batch_size = 16
n_hidden = 32
```

In [33]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [34]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [35]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

8.1.2 Architecture of LSTM

In [36]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim))) # dimensionality of the output
space=1st parameter
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 32)                5376      
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 198       
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

In [37]:

```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
              metrics=['accuracy'])
```

In [38]:

```
# Training the model
model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 43s 6ms/step - loss: 1.2997 - accuracy: 0.4619 - val_
loss: 1.0857 - val_accuracy: 0.5450
Epoch 2/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.9115 - accuracy: 0.6091 - val_
loss: 0.8464 - val_accuracy: 0.6108
Epoch 3/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.8033 - accuracy: 0.6344 - val_
loss: 0.8801 - val_accuracy: 0.5898
Epoch 4/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.6924 - accuracy: 0.6598 - val_
loss: 0.7201 - val_accuracy: 0.6111
Epoch 5/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.6595 - accuracy: 0.6733 - val_
loss: 0.7724 - val_accuracy: 0.6328
Epoch 6/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.6587 - accuracy: 0.6789 - val_
loss: 0.7239 - val_accuracy: 0.6600
Epoch 7/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.5910 - accuracy: 0.7182 - val_
loss: 0.6791 - val_accuracy: 0.7061
Epoch 8/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.5499 - accuracy: 0.7631 - val_
loss: 0.6334 - val_accuracy: 0.7431
Epoch 9/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.5075 - accuracy: 0.7817 - val_
loss: 0.5931 - val_accuracy: 0.7469
Epoch 10/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.4613 - accuracy: 0.8041 - val_
loss: 0.5119 - val_accuracy: 0.7781
Epoch 11/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.4092 - accuracy: 0.8376 - val_
loss: 0.5257 - val_accuracy: 0.8252
Epoch 12/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.3625 - accuracy: 0.8845 - val_
loss: 0.6503 - val_accuracy: 0.8029
Epoch 13/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.3071 - accuracy: 0.9082 - val_
loss: 0.4352 - val_accuracy: 0.8819
Epoch 14/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2798 - accuracy: 0.9159 - val_
loss: 0.5254 - val_accuracy: 0.8504
Epoch 15/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2652 - accuracy: 0.9246 - val_
loss: 0.4210 - val_accuracy: 0.8799
Epoch 16/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2653 - accuracy: 0.9221 - val_
loss: 0.4529 - val_accuracy: 0.8880
Epoch 17/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2163 - accuracy: 0.9355 - val_
loss: 0.3516 - val_accuracy: 0.8863
Epoch 18/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2022 - accuracy: 0.9373 - val_
loss: 0.3317 - val_accuracy: 0.8911
Epoch 19/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2006 - accuracy: 0.9406 - val_
loss: 0.3674 - val_accuracy: 0.8992
Epoch 20/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2005 - accuracy: 0.9414 - val_
loss: 0.5156 - val_accuracy: 0.8924
Epoch 21/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.2026 - accuracy: 0.9373 - val_
loss: 0.3652 - val_accuracy: 0.8989
Epoch 22/30
```

```
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1879 - accuracy: 0.9412 - val_
loss: 0.3001 - val_accuracy: 0.9118
Epoch 23/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1933 - accuracy: 0.9393 - val_
loss: 0.3761 - val_accuracy: 0.9070
Epoch 24/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1800 - accuracy: 0.9438 - val_
loss: 0.2928 - val_accuracy: 0.9094
Epoch 25/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1748 - accuracy: 0.9425 - val_
loss: 0.3346 - val_accuracy: 0.9046
Epoch 26/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1694 - accuracy: 0.9453 - val_
loss: 0.3715 - val_accuracy: 0.8962
Epoch 27/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1745 - accuracy: 0.9433 - val_
loss: 0.3248 - val_accuracy: 0.8975
Epoch 28/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1749 - accuracy: 0.9460 - val_
loss: 0.3567 - val_accuracy: 0.8992
Epoch 29/30
7352/7352 [==============================] - 41s 6ms/step - loss: 0.1672 - accuracy: 0.9461 - val_
loss: 0.2755 - val_accuracy: 0.9070
Epoch 30/30
7352/7352 [==============================] - 42s 6ms/step - loss: 0.1555 - accuracy: 0.9453 - val_
loss: 0.6149 - val_accuracy: 0.8890
```

Out[38]:

```
<keras.callbacks.callbacks.History at 0x200b8396a08>
```

In [39]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  0      379       101        4                   6
STANDING                 0       83       443        1                   1
WALKING                  0        0         0      426                  39
WALKING_DOWNSTAIRS       0        0         0        0                 418
WALKING_UPSTAIRS         0        0         0        2                  52

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            1
STANDING                           4
WALKING                           31
WALKING_DOWNSTAIRS                 2
WALKING_UPSTAIRS                 417
```

In [40]:

```
score = model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 2s 570us/step
```

In [41]:

```
score
```

Out[41]:

```
[0.6149466411874392, 0.8890396952629089]
```

8.1.3 Adding more layers in LSTM

In [42]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(32,return_sequences=True,input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))

model.add(LSTM(28,input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.6))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
dropout_2 (Dropout)          (None, 128, 32)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 28)                6832
_________________________________________________________________
dropout_3 (Dropout)          (None, 28)                0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 174
=================================================================
Total params: 12,382
Trainable params: 12,382
Non-trainable params: 0
_________________________________________________________________
```

In [43]:

```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

In [44]:

```
#Training the model
model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 84s 11ms/step - loss: 1.2226 - accuracy: 0.5175 - val
_loss: 0.8858 - val_accuracy: 0.6301
Epoch 2/30
7352/7352 [==============================] - 83s 11ms/step - loss: 0.8585 - accuracy: 0.6254 - val
_loss: 0.7991 - val_accuracy: 0.5996
Epoch 3/30
7352/7352 [==============================] - 83s 11ms/step - loss: 0.7724 - accuracy: 0.6425 - val
_loss: 0.7723 - val_accuracy: 0.6128
Epoch 4/30
7352/7352 [==============================] - 83s 11ms/step - loss: 0.7658 - accuracy: 0.6549 - val
_loss: 0.8162 - val_accuracy: 0.6759
Epoch 5/30
7352/7352 [==============================] - 83s 11ms/step - loss: 0.6941 - accuracy: 0.6859 - val
_loss: 0.7667 - val_accuracy: 0.7031
Epoch 6/30
7352/7352 [==============================] - 83s 11ms/step - loss: 0.6344 - accuracy: 0.7318 - val
_loss: 0.6183 - val_accuracy: 0.7374
Epoch 7/30
```

```
7352/7352 [==============================] - 86s 12ms/step - loss: 0.5490 - accuracy: 0.7684 - val
_loss: 0.5692 - val_accuracy: 0.7482
Epoch 8/30
7352/7352 [==============================] - 86s 12ms/step - loss: 0.4861 - accuracy: 0.7987 - val
_loss: 0.6136 - val_accuracy: 0.7676
Epoch 9/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.4500 - accuracy: 0.8043 - val
_loss: 0.5856 - val_accuracy: 0.7978
Epoch 10/30
7352/7352 [==============================] - 86s 12ms/step - loss: 0.4263 - accuracy: 0.8236 - val
_loss: 0.5595 - val_accuracy: 0.8571
Epoch 11/30
7352/7352 [==============================] - 86s 12ms/step - loss: 0.3589 - accuracy: 0.8828 - val
_loss: 0.4734 - val_accuracy: 0.8656
Epoch 12/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.3230 - accuracy: 0.9056 - val
_loss: 0.3735 - val_accuracy: 0.8843
Epoch 13/30
7352/7352 [==============================] - 87s 12ms/step - loss: 0.2771 - accuracy: 0.9187 - val
_loss: 0.3720 - val_accuracy: 0.8907
Epoch 14/30
7352/7352 [==============================] - 86s 12ms/step - loss: 0.2661 - accuracy: 0.9210 - val
_loss: 0.3748 - val_accuracy: 0.8948
Epoch 15/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.2333 - accuracy: 0.9335 - val
_loss: 0.4607 - val_accuracy: 0.8860
Epoch 16/30
7352/7352 [==============================] - 86s 12ms/step - loss: 0.2288 - accuracy: 0.9317 - val
_loss: 0.4430 - val_accuracy: 0.8802
Epoch 17/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.2010 - accuracy: 0.9387 - val
_loss: 0.3360 - val_accuracy: 0.9074
Epoch 18/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.2031 - accuracy: 0.9384 - val
_loss: 0.4535 - val_accuracy: 0.8948
Epoch 19/30
7352/7352 [==============================] - 84s 11ms/step - loss: 0.1969 - accuracy: 0.9449 - val
_loss: 0.4181 - val_accuracy: 0.9026
Epoch 20/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.2043 - accuracy: 0.9393 - val
_loss: 0.4076 - val_accuracy: 0.9013
Epoch 21/30
7352/7352 [==============================] - 86s 12ms/step - loss: 0.2051 - accuracy: 0.9403 - val
_loss: 0.5088 - val_accuracy: 0.8955
Epoch 22/30
7352/7352 [==============================] - 84s 11ms/step - loss: 0.1769 - accuracy: 0.9450 - val
_loss: 0.4067 - val_accuracy: 0.9033
Epoch 23/30
7352/7352 [==============================] - 84s 11ms/step - loss: 0.1869 - accuracy: 0.9425 - val
_loss: 0.4893 - val_accuracy: 0.8965
Epoch 24/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.1781 - accuracy: 0.9425 - val
_loss: 0.4413 - val_accuracy: 0.9013
Epoch 25/30
7352/7352 [==============================] - 84s 11ms/step - loss: 0.1800 - accuracy: 0.9501 - val
_loss: 0.6307 - val_accuracy: 0.9077accuracy: 0.
Epoch 26/30
7352/7352 [==============================] - 85s 11ms/step - loss: 0.1776 - accuracy: 0.9436 - val
_loss: 0.5969 - val_accuracy: 0.9053
Epoch 27/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.1690 - accuracy: 0.9464 - val
_loss: 0.5011 - val_accuracy: 0.9046
Epoch 28/30
7352/7352 [==============================] - 84s 11ms/step - loss: 0.1891 - accuracy: 0.9429 - val
_loss: 0.4572 - val_accuracy: 0.9013
Epoch 29/30
7352/7352 [==============================] - 84s 11ms/step - loss: 0.1835 - accuracy: 0.9438 - val
_loss: 0.4119 - val_accuracy: 0.8962
Epoch 30/30
7352/7352 [==============================] - 85s 12ms/step - loss: 0.1752 - accuracy: 0.9422 - val
_loss: 0.5288 - val_accuracy: 0.9013
```

Out[44]:

```
<keras.callbacks.callbacks.History at 0x200b8297448>
```

1)90.13% Accuracy

2)1.23% increase in Accuracy between 1 Layer of LSTM and 2 Layers of LSTM.

3)The Train loss and the validation loss has a huge difference, so now we add L2 regularization to minimize the error.

recurrent_regularization() https://keras.io/api/layers/recurrent_layers/lstm/

In [45]:

```
from keras.regularizers import l2
model = Sequential()
# Configuring the parameters
model.add(LSTM(32,recurrent_regularizer=l2(0.003),return_sequences=True,input_shape=(timesteps,
input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))

model.add(LSTM(28,input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.6))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate
instead of keep_prob. Please ensure that this is intended.
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 128, 32)           5376
_________________________________________________________________
dropout_4 (Dropout)          (None, 128, 32)           0
_________________________________________________________________
lstm_5 (LSTM)                (None, 28)                6832
_________________________________________________________________
dropout_5 (Dropout)          (None, 28)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 174
=================================================================
Total params: 12,382
Trainable params: 12,382
Non-trainable params: 0
_________________________________________________________________
```

In [46]:

```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

In [47]:

```
# Training the model
History = model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=10)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/10
7352/7352 [==============================] - 86s 12ms/step - loss: 1.4069 - accuracy: 0.4047 - val
_loss: 1.1523 - val_accuracy: 0.5745
Epoch 2/10
7352/7352 [==============================] - 86s 12ms/step - loss: 0.9823 - accuracy: 0.5865 - val
_loss: 0.8636 - val_accuracy: 0.5989
Epoch 3/10
7352/7352 [==============================] - 86s 12ms/step - loss: 0.8136 - accuracy: 0.6314 - val
_loss: 0.7569 - val_accuracy: 0.6288
```

```
Epoch 4/10
7352/7352 [==============================] - 86s 12ms/step - loss: 0.8333 - accuracy: 0.6386 - val
_loss: 0.8920 - val_accuracy: 0.5969
Epoch 5/10
7352/7352 [==============================] - 85s 12ms/step - loss: 0.7802 - accuracy: 0.6447 - val
_loss: 0.7715 - val_accuracy: 0.6196
Epoch 6/10
7352/7352 [==============================] - 85s 12ms/step - loss: 0.7550 - accuracy: 0.6541 - val
_loss: 0.7418 - val_accuracy: 0.6203
Epoch 7/10
7352/7352 [==============================] - 85s 12ms/step - loss: 0.7188 - accuracy: 0.6576 - val
_loss: 0.7182 - val_accuracy: 0.6810
Epoch 8/10
7352/7352 [==============================] - 85s 12ms/step - loss: 0.6586 - accuracy: 0.7006 - val
_loss: 0.6561 - val_accuracy: 0.6848
Epoch 9/10
7352/7352 [==============================] - 85s 12ms/step - loss: 0.5920 - accuracy: 0.7533 - val
_loss: 0.6094 - val_accuracy: 0.7424
Epoch 10/10
7352/7352 [==============================] - 85s 12ms/step - loss: 0.5358 - accuracy: 0.8070 - val
_loss: 0.6374 - val_accuracy: 0.7991
```

Accuracy has dropped to 79.91%

8.1.4 Hyper-Parameter Tuning with Hyperas and Applying LSTM with best Hyper-Parameters

In [49]:

```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
from hyperopt import Trials, STATUS_OK, tpe
from hyperas import optim
from hyperas.distributions import choice, uniform
from hyperas.utils import eval_hyperopt_space
#https://github.com/UdiBhaskar/Human-Activity-Recognition--Using-Deep-
NN/blob/master/Human%20Activity%20Detection.ipynb
```

In [37]:

```
##gives train and validation data
def data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    # Data directory
    DATADIR = 'UCI_HAR_Dataset'
    # Raw data signals
    # Signals are from Accelerometer and Gyroscope
    # The signals are in x,y,z directions
    # Sensor signals are filtered to have only body acceleration
    # excluding the acceleration due to gravity
    # Triaxial acceleration from the accelerometer is total acceleration
    SIGNALS = [
        "body_acc_x",
        "body_acc_y",
        "body_acc_z",
        "body_gyro_x",
        "body_gyro_y",
        "body_gyro_z",
        "total_acc_x",
        "total_acc_y",
        "total_acc_z"
        ]
    # Utility function to read the data from csv file
    def _read_csv(filename):
        return pd.read_csv(filename, delim_whitespace=True, header=None,encoding='utf-8')

    # Utility function to load the load
    def load_signals(subset):
```

```
        signals_data = []

        for signal in SIGNALS:
            filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
            signals_data.append( _read_csv(filename).values)

        # Transpose is used to change the dimensionality of the output,
        # aggregating the signals by combination of sample/timestep.
        # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
        return np.transpose(signals_data, (1, 2, 0))

    def load_y(subset):
        """
        The objective that we are trying to predict is a integer, from 1 to 6,
        that represents a human activity. We return a binary representation of
        every sample objective as a 6 bits vector using One Hot Encoding
        (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
        """
        filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
        y = _read_csv(filename)[0]
        return pd.get_dummies(y).values

    X_train, X_val = load_signals('train'), load_signals('test')
    Y_train, Y_val = load_y('train'), load_y('test')

    return X_train, Y_train, X_val,  Y_val
```

In [50]:

```
from keras.regularizers import l2
import keras

##model
def model(X_train, Y_train, X_val, Y_val):
    # Importing tensorflow
    np.random.seed(36)
    import tensorflow as tf
    #tf.set_random_seed(36)
    tf.random.set_seed(36)
    # Initiliazing the sequential model
    model = Sequential()
    # choice==============>Gives out one of the input options randomly as output
    if ({{choice(['one', 'two'])}}) == 'two':
        # Configuring the parameters
        # choice function (selector, selection)=======> A choice function (selector, selection) is
a mathematical function f that is defined on some collection X of nonempty sets and assigns to eac
h set S in that collection some element f(S) of S. In other words, f is a choice function for X if
and only if it belongs to the direct product of X.
        # uniform() =============>returns a random floating-point number between a given range of
numbers
        model.add(LSTM({{choice([28,32,38])}},recurrent_regularizer=l2({{uniform(0,0.0002)}}),retur
n_sequences=True,input_shape=(128, 9),name='LSTM2_1'))
        # Adding a dropout layer
        model.add(Dropout({{uniform(0.35,0.65)}},name='Dropout2_1'))
        model.add(LSTM({{choice([26,32,36])}},recurrent_regularizer=l2({{uniform(0,0.001)}}),input_
shape=(128, 9),name='LSTM2_2'))
        model.add(Dropout({{uniform(0.5,0.7)}},name='Dropout2_2'))
        # Adding a dense output layer with sigmoid activation
        model.add(Dense(6, activation='sigmoid'))
    else:
        # Configuring the parameters
        model.add(LSTM({{choice([28,32,36])}},recurrent_regularizer=l2({{uniform(0,0.001)}}),input_
shape=(128, 9),name='LSTM1_1'))
        # Adding a dropout layer
        model.add(Dropout({{uniform(0.35,0.55)}},name='Dropout1_1'))
        # Adding a dense output layer with sigmoid activation
        model.add(Dense(6, activation='sigmoid'))

    adam = keras.optimizers.Adam(lr={{uniform(0.009,0.025)}})
    rmsprop = keras.optimizers.RMSprop(lr={{uniform(0.009,0.025)}})

    choiceval = {{choice(['adam', 'rmsprop'])}}

    if choiceval == 'adam':
        optim = adam
    else:
        optim = rmsprop
```

```
        optim = rmsprop

    print(model.summary())

    model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer=optim)

    result = model.fit(X_train, Y_train,
              batch_size=16,
              nb_epoch=30,
              verbose=2,
              validation_data=(X_val, Y_val))

    score, acc = model.evaluate(X_val, Y_val, verbose=0)
    print('Test accuracy:', acc)
    print('-------------------------------------------------------------------------------------')
    return {'loss': -acc, 'status': STATUS_OK, 'model': model}
```

In [51]:

```
X_train, Y_train, X_val, Y_val = data()
trials = Trials()
best_run, best_model, space = optim.minimize(model=model,
                                      data=data,
                                      algo=tpe.suggest,
                                      max_evals=15,
                                      trials=trials,notebook_name = 'Untitled',
                                     return_space = True)
```

```
>>> Imports:
#coding=utf-8

try:
    import numpy as np
except:
    pass

try:
    import pandas as pd
except:
    pass

try:
    import seaborn as sns
except:
    pass

try:
    import matplotlib.pyplot as plt
except:
    pass

try:
    import tensorflow as tf
except:
    pass

try:
    from tensorflow.python.keras import backend as K
except:
    pass

try:
    from keras.models import Sequential
except:
    pass

try:
    import itertools
except:
    pass

try:
    import numpy as np
except:
    pass
```

```
try:
    import matplotlib.pyplot as plt
except:
    pass

try:
    from sklearn.metrics import confusion_matrix
except:
    pass

try:
    from datetime import datetime
except:
    pass

try:
    from sklearn import linear_model
except:
    pass

try:
    from sklearn import metrics
except:
    pass

try:
    from sklearn.model_selection import GridSearchCV
except:
    pass

try:
    from sklearn.svm import LinearSVC
except:
    pass

try:
    from sklearn.svm import SVC
except:
    pass

try:
    from sklearn.tree import DecisionTreeClassifier
except:
    pass

try:
    from sklearn.ensemble import RandomForestClassifier
except:
    pass

try:
    from sklearn.ensemble import GradientBoostingClassifier
except:
    pass

try:
    import tensorflow as tf
except:
    pass

try:
    from tensorflow.python.keras import backend as K
except:
    pass

try:
    from keras.models import Sequential
except:
    pass

try:
    from keras.layers import LSTM
except:
    pass

try:
    from keras.layers.core import Dense, Dropout
```

```
    from keras.layers.core import Dense, Dropout
except:
    pass

try:
    from keras.regularizers import l2
except:
    pass

try:
    from keras.models import Sequential
except:
    pass

try:
    from keras.layers import LSTM
except:
    pass

try:
    from keras.layers.core import Dense, Dropout
except:
    pass

try:
    from hyperopt import Trials, STATUS_OK, tpe
except:
    pass

try:
    from hyperas import optim
except:
    pass

try:
    from hyperas.distributions import choice, uniform
except:
    pass

try:
    from hyperas.utils import eval_hyperopt_space
except:
    pass

try:
    from keras.regularizers import l2
except:
    pass

try:
    import keras
except:
    pass

try:
    import tensorflow as tf
except:
    pass

try:
    import sklearn.metrics as metrics
except:
    pass

try:
    import os
except:
    pass

try:
    import random as rn
except:
    pass

try:
    from sklearn.preprocessing import StandardScaler
except:
    pass
```

```
    pass

try:
    from sklearn.base import BaseEstimator, TransformerMixin
except:
    pass

try:
    from keras.layers.convolutional import Conv1D
except:
    pass

try:
    from keras.layers.convolutional import MaxPooling1D
except:
    pass

try:
    from keras.utils import to_categorical
except:
    pass

try:
    from keras.layers import Flatten
except:
    pass

try:
    import math
except:
    pass

try:
    from keras.callbacks import LearningRateScheduler
except:
    pass

>>> Hyperas search space:

def get_space():
    return {
        'if': hp.choice('if', ['one', 'two']),
        'LSTM': hp.choice('LSTM', [28,32,38]),
        'l2': hp.uniform('l2', 0,0.0002),
        'Dropout': hp.uniform('Dropout', 0.35,0.65),
        'LSTM_1': hp.choice('LSTM_1', [26,32,36]),
        'l2_1': hp.uniform('l2_1', 0,0.001),
        'Dropout_1': hp.uniform('Dropout_1', 0.5,0.7),
        'LSTM_2': hp.choice('LSTM_2', [28,32,36]),
        'l2_2': hp.uniform('l2_2', 0,0.001),
        'Dropout_2': hp.uniform('Dropout_2', 0.35,0.55),
        'lr': hp.uniform('lr', 0.009,0.025),
        'lr_1': hp.uniform('lr_1', 0.009,0.025),
        'choiceval': hp.choice('choiceval', ['adam', 'rmsprop']),
    }

>>> Data

"""
Obtain the dataset from multiple files.
Returns: X_train, X_test, y_train, y_test
"""
# Data directory
DATADIR = 'UCI_HAR_Dataset'
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
```

```
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
    ]
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None,encoding='utf-8')

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append( _read_csv(filename).values)

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))

def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]
    return pd.get_dummies(y).values

X_train, X_val = load_signals('train'), load_signals('test')
Y_train, Y_val = load_y('train'), load_y('test')




```

>>> Resulting replaced keras model:

```
def keras_fmin_fnct(space):

    # Importing tensorflow
    np.random.seed(36)
    #tf.set_random_seed(36)
    tf.random.set_seed(36)
    # Initiliazing the sequential model
    model = Sequential()
    if (space['if']) == 'two':
        # Configuring the parameters
model.add(LSTM(space['LSTM'],recurrent_regularizer=l2(space['l2']),return_sequences=True,input_shap
(128, 9),name='LSTM2_1'))
        # Adding a dropout layer
        model.add(Dropout(space['Dropout'],name='Dropout2_1'))
        model.add(LSTM(space['LSTM_1'],recurrent_regularizer=l2(space['l2_1']),input_shape=(128, 9),name='LSTM2_2'))
        model.add(Dropout(space['Dropout_1'],name='Dropout2_2'))
        # Adding a dense output layer with sigmoid activation
        model.add(Dense(6, activation='sigmoid'))
    else:
        # Configuring the parameters
        model.add(LSTM(space['LSTM_2'],recurrent_regularizer=l2(space['l2_2']),input_shape=(128, 9),name='LSTM1_1'))
        # Adding a dropout layer
        model.add(Dropout(space['Dropout_2'],name='Dropout1_1'))
        # Adding a dense output layer with sigmoid activation
        model.add(Dense(6, activation='sigmoid'))

    adam = keras.optimizers.Adam(lr=space['lr'])
    rmsprop = keras.optimizers.RMSprop(lr=space['lr_1'])

    choiceval = space['choiceval']

    if choiceval == 'adam':
        optim = adam
    else:
        optim = rmsprop
```

```
        optim = rmsprop

    print(model.summary())

    model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer=optim)

    result = model.fit(X_train, Y_train,
              batch_size=16,
              nb_epoch=30,
              verbose=2,
              validation_data=(X_val, Y_val))

    score, acc = model.evaluate(X_val, Y_val, verbose=0)
    print('Test accuracy:', acc)
    print('-----------------------------------------------------------------------------------
')
    return {'loss': -acc, 'status': STATUS_OK, 'model': model}
```

```
  0%|                                                                            | 0/15 [00:00<?, ?t
l/s, best loss=?]WARNING:tensorflow:Large dropout rate: 0.518883 (>0.5). In TensorFlow 2.x,
dropout() uses dropout rate instead of keep_prob. Please ensure that this is intended.
Model: "sequential_4"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM1_1 (LSTM)               (None, 32)                5376

_________________________________________________________________

Dropout1_1 (Dropout)         (None, 32)                0

_________________________________________________________________

dense_4 (Dense)              (None, 6)                 198

=================================================================

Total params: 5,574

Trainable params: 5,574

Non-trainable params: 0

_________________________________________________________________

None

  0%|                                                                            | 0/15 [00:00<?, ?t
l/s, best loss=?]
```

```
C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: T
he `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 41s - loss: 1.0128 - accuracy: 0.5547 - val_loss: 0.7615 - val_accuracy: 0.6434


Epoch 2/30

 - 41s - loss: 0.6721 - accuracy: 0.7114 - val_loss: 0.6198 - val_accuracy: 0.7567


Epoch 3/30

 - 41s - loss: 0.4823 - accuracy: 0.8554 - val_loss: 0.4824 - val_accuracy: 0.8612


Epoch 4/30
```

```
 - 41s - loss: 0.3744 - accuracy: 0.9066 - val_loss: 0.4864 - val_accuracy: 0.8809

Epoch 5/30

 - 41s - loss: 0.2886 - accuracy: 0.9223 - val_loss: 0.3887 - val_accuracy: 0.8806

Epoch 6/30

 - 41s - loss: 0.2770 - accuracy: 0.9256 - val_loss: 0.4971 - val_accuracy: 0.8741

Epoch 7/30

 - 41s - loss: 0.2404 - accuracy: 0.9335 - val_loss: 0.5900 - val_accuracy: 0.8823

Epoch 8/30

 - 41s - loss: 0.2534 - accuracy: 0.9332 - val_loss: 0.3340 - val_accuracy: 0.9121

Epoch 9/30

 - 41s - loss: 0.2286 - accuracy: 0.9368 - val_loss: 0.4357 - val_accuracy: 0.9094

Epoch 10/30

 - 41s - loss: 0.2260 - accuracy: 0.9370 - val_loss: 0.3744 - val_accuracy: 0.9165

Epoch 11/30

 - 42s - loss: 0.2038 - accuracy: 0.9402 - val_loss: 0.3447 - val_accuracy: 0.9067

Epoch 12/30

 - 41s - loss: 0.2044 - accuracy: 0.9433 - val_loss: 0.3276 - val_accuracy: 0.9277

Epoch 13/30

 - 41s - loss: 0.1964 - accuracy: 0.9402 - val_loss: 0.4485 - val_accuracy: 0.8823

Epoch 14/30

 - 41s - loss: 0.1961 - accuracy: 0.9410 - val_loss: 0.5780 - val_accuracy: 0.8968

Epoch 15/30

 - 41s - loss: 0.1829 - accuracy: 0.9415 - val_loss: 0.5610 - val_accuracy: 0.8941

Epoch 16/30

 - 41s - loss: 0.1923 - accuracy: 0.9403 - val_loss: 0.5484 - val_accuracy: 0.8948

Epoch 17/30

 - 41s - loss: 0.1967 - accuracy: 0.9399 - val_loss: 0.4251 - val_accuracy: 0.8843

Epoch 18/30

 - 41s - loss: 0.1870 - accuracy: 0.9425 - val_loss: 0.5657 - val_accuracy: 0.8877

Epoch 19/30

 - 41s - loss: 0.1799 - accuracy: 0.9416 - val_loss: 0.4302 - val_accuracy: 0.9067
```

```
Epoch 20/30

 - 41s - loss: 0.1851 - accuracy: 0.9456 - val_loss: 0.3678 - val_accuracy: 0.8982

Epoch 21/30

 - 42s - loss: 0.1820 - accuracy: 0.9407 - val_loss: 0.5228 - val_accuracy: 0.8914

Epoch 22/30

 - 42s - loss: 0.1717 - accuracy: 0.9440 - val_loss: 0.5489 - val_accuracy: 0.8836

Epoch 23/30

 - 41s - loss: 0.1930 - accuracy: 0.9411 - val_loss: 0.3631 - val_accuracy: 0.8958

Epoch 24/30

 - 42s - loss: 0.1792 - accuracy: 0.9425 - val_loss: 0.5854 - val_accuracy: 0.8884

Epoch 25/30

 - 41s - loss: 0.1696 - accuracy: 0.9452 - val_loss: 0.5340 - val_accuracy: 0.8694

Epoch 26/30

 - 41s - loss: 0.1749 - accuracy: 0.9442 - val_loss: 0.4755 - val_accuracy: 0.8887

Epoch 27/30

 - 41s - loss: 0.1898 - accuracy: 0.9402 - val_loss: 0.4342 - val_accuracy: 0.8931

Epoch 28/30

 - 41s - loss: 0.1703 - accuracy: 0.9459 - val_loss: 0.4731 - val_accuracy: 0.9026

Epoch 29/30

 - 41s - loss: 0.1723 - accuracy: 0.9465 - val_loss: 0.4159 - val_accuracy: 0.9057

Epoch 30/30

 - 41s - loss: 0.1623 - accuracy: 0.9514 - val_loss: 0.5049 - val_accuracy: 0.9070

Test accuracy:

0.907024085521698

--------------------------------------------------------------------------------------

  7%|███                                         | 1/15 [20:39<4:49:10, 1239.34s/trial, best loss:
-0.907024085521698]WARNING:tensorflow:Large dropout rate: 0.604072 (>0.5). In TensorFlow 2.x,
dropout() uses dropout rate instead of keep_prob. Please ensure that this is intended.
WARNING:tensorflow:Large dropout rate: 0.564208 (>0.5). In TensorFlow 2.x, dropout() uses dropout
rate instead of keep_prob. Please ensure that this is intended.
Model: "sequential_5"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 28)           4256
```

```
_________________________________________________________________
Dropout2_1 (Dropout)         (None, 128, 28)          0
_________________________________________________________________
LSTM2_2 (LSTM)               (None, 32)               7808
_________________________________________________________________
Dropout2_2 (Dropout)         (None, 32)               0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                198
=================================================================
Total params: 12,262
Trainable params: 12,262
Non-trainable params: 0
_________________________________________________________________
None
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 84s - loss: 1.2701 - accuracy: 0.4423 - val_loss: 1.1434 - val_accuracy: 0.4805

Epoch 2/30
 - 84s - loss: 0.9939 - accuracy: 0.5558 - val_loss: 0.9179 - val_accuracy: 0.5891

Epoch 3/30
 - 84s - loss: 0.7789 - accuracy: 0.6216 - val_loss: 0.8226 - val_accuracy: 0.6047

Epoch 4/30
 - 84s - loss: 0.7716 - accuracy: 0.6345 - val_loss: 0.9260 - val_accuracy: 0.6006

Epoch 5/30
 - 84s - loss: 0.7385 - accuracy: 0.6483 - val_loss: 0.8447 - val_accuracy: 0.6281

Epoch 6/30
 - 84s - loss: 0.6856 - accuracy: 0.6683 - val_loss: 0.7739 - val_accuracy: 0.6308

Epoch 7/30
 - 83s - loss: 0.5678 - accuracy: 0.7926 - val_loss: 0.8544 - val_accuracy: 0.8107

Epoch 8/30
 - 83s - loss: 0.3767 - accuracy: 0.9008 - val_loss: 0.7246 - val_accuracy: 0.8500

Epoch 9/30
 - 84s - loss: 0.3730 - accuracy: 0.9094 - val_loss: 0.7592 - val_accuracy: 0.8775

Epoch 10/30
```

```
 - 83s - loss: 0.3250 - accuracy: 0.9219 - val_loss: 0.7527 - val_accuracy: 0.8812

Epoch 11/30

 - 84s - loss: 0.2631 - accuracy: 0.9353 - val_loss: 0.6969 - val_accuracy: 0.8643

Epoch 12/30

 - 84s - loss: 0.2535 - accuracy: 0.9289 - val_loss: 0.8298 - val_accuracy: 0.8806

Epoch 13/30

 - 83s - loss: 0.2522 - accuracy: 0.9358 - val_loss: 0.6422 - val_accuracy: 0.8809

Epoch 14/30

 - 84s - loss: 0.2435 - accuracy: 0.9412 - val_loss: 0.6664 - val_accuracy: 0.8921

Epoch 15/30

 - 81s - loss: 0.2165 - accuracy: 0.9434 - val_loss: 0.6484 - val_accuracy: 0.8853

Epoch 16/30

 - 82s - loss: 0.2119 - accuracy: 0.9384 - val_loss: 0.8834 - val_accuracy: 0.8839

Epoch 17/30

 - 82s - loss: 0.2131 - accuracy: 0.9442 - val_loss: 0.8392 - val_accuracy: 0.8951

Epoch 18/30

 - 81s - loss: 0.2699 - accuracy: 0.9350 - val_loss: 0.9180 - val_accuracy: 0.8907

Epoch 19/30

 - 82s - loss: 0.2204 - accuracy: 0.9429 - val_loss: 0.9846 - val_accuracy: 0.8897

Epoch 20/30

 - 82s - loss: 0.2356 - accuracy: 0.9389 - val_loss: 0.7360 - val_accuracy: 0.9074

Epoch 21/30

 - 83s - loss: 0.2618 - accuracy: 0.9387 - val_loss: 0.9494 - val_accuracy: 0.8907

Epoch 22/30

 - 83s - loss: 0.2150 - accuracy: 0.9397 - val_loss: 0.8550 - val_accuracy: 0.8962

Epoch 23/30

 - 83s - loss: 0.1898 - accuracy: 0.9410 - val_loss: 0.9267 - val_accuracy: 0.8829

Epoch 24/30

 - 86s - loss: 0.1954 - accuracy: 0.9422 - val_loss: 0.9673 - val_accuracy: 0.8945

Epoch 25/30

 - 84s - loss: 0.2068 - accuracy: 0.9434 - val_loss: 0.9386 - val_accuracy: 0.8836
```

```
Epoch 26/30

 - 90s - loss: 0.1694 - accuracy: 0.9475 - val_loss: 0.9369 - val_accuracy: 0.8711

Epoch 27/30

 - 86s - loss: 0.3627 - accuracy: 0.9263 - val_loss: 1.0849 - val_accuracy: 0.8772

Epoch 28/30

 - 89s - loss: 0.3798 - accuracy: 0.9290 - val_loss: 1.0810 - val_accuracy: 0.8778

Epoch 29/30

 - 85s - loss: 0.1988 - accuracy: 0.9421 - val_loss: 1.1714 - val_accuracy: 0.8755

Epoch 30/30

 - 84s - loss: 0.1997 - accuracy: 0.9465 - val_loss: 0.9991 - val_accuracy: 0.8755

Test accuracy:

0.8754665851593018

--------------------------------------------------------------------------------------

Model: "sequential_6"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 38)           7296

_________________________________________________________________

Dropout2_1 (Dropout)         (None, 128, 38)           0

_________________________________________________________________

LSTM2_2 (LSTM)               (None, 36)                10800

_________________________________________________________________

Dropout2_2 (Dropout)         (None, 36)                0

_________________________________________________________________

dense_6 (Dense)              (None, 6)                 222

=================================================================

Total params: 18,318

Trainable params: 18,318

Non-trainable params: 0

_________________________________________________________________

None

 13%|██████                                     | 2/15 [1:02:38<5:51:39, 1623.04s/trial, best loss: -0.907024085521698]

C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 86s - loss: 1.3083 - accuracy: 0.4237 - val_loss: 1.0261 - val_accuracy: 0.5860

Epoch 2/30
 - 86s - loss: 0.8399 - accuracy: 0.6235 - val_loss: 0.8142 - val_accuracy: 0.6240

Epoch 3/30
 - 87s - loss: 0.7639 - accuracy: 0.6451 - val_loss: 0.8486 - val_accuracy: 0.6216

Epoch 4/30
 - 88s - loss: 0.5857 - accuracy: 0.7658 - val_loss: 0.5934 - val_accuracy: 0.7903

Epoch 5/30
 - 87s - loss: 0.4618 - accuracy: 0.8628 - val_loss: 0.4518 - val_accuracy: 0.8982

Epoch 6/30
 - 89s - loss: 0.3715 - accuracy: 0.9119 - val_loss: 0.4176 - val_accuracy: 0.8972

Epoch 7/30
 - 87s - loss: 0.3123 - accuracy: 0.9212 - val_loss: 0.5936 - val_accuracy: 0.8680

Epoch 8/30
 - 88s - loss: 0.2886 - accuracy: 0.9253 - val_loss: 0.5456 - val_accuracy: 0.8829

Epoch 9/30
 - 86s - loss: 0.2513 - accuracy: 0.9331 - val_loss: 0.6375 - val_accuracy: 0.8918

Epoch 10/30
 - 85s - loss: 0.2606 - accuracy: 0.9321 - val_loss: 0.5425 - val_accuracy: 0.9087

Epoch 11/30
 - 84s - loss: 0.2631 - accuracy: 0.9369 - val_loss: 0.4584 - val_accuracy: 0.9026

Epoch 12/30
 - 85s - loss: 0.2447 - accuracy: 0.9354 - val_loss: 0.5095 - val_accuracy: 0.9128

Epoch 13/30
 - 85s - loss: 0.2435 - accuracy: 0.9359 - val_loss: 0.5281 - val_accuracy: 0.9067

Epoch 14/30
 - 85s - loss: 0.2471 - accuracy: 0.9310 - val_loss: 0.7812 - val_accuracy: 0.9046

Epoch 15/30
 - 86s - loss: 0.2147 - accuracy: 0.9385 - val_loss: 0.5709 - val_accuracy: 0.9002
```

```
 - 86s - loss: 0.2147 - accuracy: 0.9385 - val_loss: 0.5709 - val_accuracy: 0.9002

Epoch 16/30

 - 87s - loss: 0.2002 - accuracy: 0.9421 - val_loss: 0.6614 - val_accuracy: 0.9097

Epoch 17/30

 - 85s - loss: 0.1909 - accuracy: 0.9408 - val_loss: 0.5294 - val_accuracy: 0.9074

Epoch 18/30

 - 86s - loss: 0.2178 - accuracy: 0.9400 - val_loss: 0.5210 - val_accuracy: 0.8996

Epoch 19/30

 - 86s - loss: 0.2002 - accuracy: 0.9397 - val_loss: 0.5327 - val_accuracy: 0.8911

Epoch 20/30

 - 85s - loss: 0.1787 - accuracy: 0.9415 - val_loss: 0.6870 - val_accuracy: 0.9060

Epoch 21/30

 - 84s - loss: 0.1999 - accuracy: 0.9416 - val_loss: 0.6837 - val_accuracy: 0.9067

Epoch 22/30

 - 84s - loss: 0.1874 - accuracy: 0.9452 - val_loss: 0.4706 - val_accuracy: 0.9094

Epoch 23/30

 - 84s - loss: 0.1749 - accuracy: 0.9422 - val_loss: 0.5920 - val_accuracy: 0.9111

Epoch 24/30

 - 87s - loss: 0.1856 - accuracy: 0.9448 - val_loss: 0.5089 - val_accuracy: 0.9169

Epoch 25/30

 - 84s - loss: 0.1641 - accuracy: 0.9453 - val_loss: 0.6518 - val_accuracy: 0.9013

Epoch 26/30

 - 85s - loss: 0.1884 - accuracy: 0.9449 - val_loss: 0.7065 - val_accuracy: 0.8948

Epoch 27/30

 - 90s - loss: 0.1714 - accuracy: 0.9474 - val_loss: 0.5691 - val_accuracy: 0.9165

Epoch 28/30

 - 91s - loss: 0.1669 - accuracy: 0.9448 - val_loss: 0.4775 - val_accuracy: 0.9257

Epoch 29/30

 - 86s - loss: 0.1687 - accuracy: 0.9455 - val_loss: 0.7080 - val_accuracy: 0.8965

Epoch 30/30

 - 84s - loss: 0.1933 - accuracy: 0.9452 - val_loss: 0.7150 - val_accuracy: 0.8996
```

```
Test accuracy:

0.8995589017868042

--------------------------------------------------------------------------------------

Model: "sequential_7"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 32)           5376

_________________________________________________________________

Dropout2_1 (Dropout)         (None, 128, 32)           0

_________________________________________________________________

LSTM2_2 (LSTM)               (None, 32)                8320

_________________________________________________________________

Dropout2_2 (Dropout)         (None, 32)                0

_________________________________________________________________

dense_7 (Dense)              (None, 6)                 198

=================================================================

Total params: 13,894

Trainable params: 13,894

Non-trainable params: 0

_________________________________________________________________

None

 20%|█████████                                    | 3/15 [1:45:46<6:22:33, 1912.83s/trial, best loss:
-0.907024085521698]
```

```
C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: T
he `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 87s - loss: 1.3970 - accuracy: 0.3640 - val_loss: 1.4054 - val_accuracy: 0.3536


Epoch 2/30

 - 87s - loss: 1.3703 - accuracy: 0.3675 - val_loss: 1.4372 - val_accuracy: 0.3536


Epoch 3/30

 - 86s - loss: 1.3628 - accuracy: 0.3878 - val_loss: 1.3802 - val_accuracy: 0.3539


Epoch 4/30

 - 86s - loss: 1.4661 - accuracy: 0.3761 - val_loss: 1.6313 - val_accuracy: 0.4394


Epoch 5/30
```

```
 - 87s - loss: 1.5168 - accuracy: 0.4271 - val_loss: 1.2960 - val_accuracy: 0.4958

Epoch 6/30

 - 86s - loss: 1.2957 - accuracy: 0.4950 - val_loss: 1.1213 - val_accuracy: 0.5239

Epoch 7/30

 - 86s - loss: 1.1322 - accuracy: 0.5413 - val_loss: 1.1111 - val_accuracy: 0.5850

Epoch 8/30

 - 86s - loss: 1.1523 - accuracy: 0.5292 - val_loss: 1.1000 - val_accuracy: 0.5042

Epoch 9/30

 - 87s - loss: 1.0539 - accuracy: 0.5409 - val_loss: 1.0364 - val_accuracy: 0.5239

Epoch 10/30

 - 86s - loss: 0.9852 - accuracy: 0.5973 - val_loss: 0.9880 - val_accuracy: 0.5843

Epoch 11/30

 - 86s - loss: 0.8926 - accuracy: 0.6455 - val_loss: 0.9300 - val_accuracy: 0.5935

Epoch 12/30

 - 86s - loss: 0.8783 - accuracy: 0.6499 - val_loss: 0.9931 - val_accuracy: 0.6159

Epoch 13/30

 - 86s - loss: 0.8880 - accuracy: 0.6548 - val_loss: 1.0337 - val_accuracy: 0.5582

Epoch 14/30

 - 85s - loss: 0.8444 - accuracy: 0.6476 - val_loss: 0.9673 - val_accuracy: 0.6149

Epoch 15/30

 - 87s - loss: 0.8061 - accuracy: 0.6575 - val_loss: 0.9389 - val_accuracy: 0.6216

Epoch 16/30

 - 87s - loss: 0.7603 - accuracy: 0.6619 - val_loss: 0.9618 - val_accuracy: 0.5813

Epoch 17/30

 - 88s - loss: 0.7756 - accuracy: 0.6492 - val_loss: 0.9378 - val_accuracy: 0.6003

Epoch 18/30

 - 88s - loss: 0.7413 - accuracy: 0.6608 - val_loss: 0.8773 - val_accuracy: 0.6261

Epoch 19/30

 - 88s - loss: 0.7152 - accuracy: 0.6649 - val_loss: 0.8964 - val_accuracy: 0.6159

Epoch 20/30

 - 88s - loss: 0.7418 - accuracy: 0.6604 - val_loss: 0.7896 - val_accuracy: 0.6373
```

```
Epoch 21/30

 - 89s - loss: 0.7241 - accuracy: 0.6549 - val_loss: 0.8153 - val_accuracy: 0.6288

Epoch 22/30

 - 89s - loss: 0.7228 - accuracy: 0.6604 - val_loss: 0.7355 - val_accuracy: 0.6393

Epoch 23/30

 - 89s - loss: 0.6842 - accuracy: 0.6696 - val_loss: 0.7590 - val_accuracy: 0.6454

Epoch 24/30

 - 90s - loss: 0.6826 - accuracy: 0.6625 - val_loss: 0.7266 - val_accuracy: 0.6675

Epoch 25/30

 - 89s - loss: 0.6951 - accuracy: 0.6649 - val_loss: 0.7230 - val_accuracy: 0.6383

Epoch 26/30

 - 89s - loss: 0.6856 - accuracy: 0.6702 - val_loss: 0.7358 - val_accuracy: 0.6437

Epoch 27/30

 - 90s - loss: 0.6643 - accuracy: 0.6710 - val_loss: 0.7028 - val_accuracy: 0.6502

Epoch 28/30

 - 90s - loss: 0.6906 - accuracy: 0.6670 - val_loss: 0.7016 - val_accuracy: 0.6315

Epoch 29/30

 - 90s - loss: 0.6511 - accuracy: 0.6717 - val_loss: 0.7010 - val_accuracy: 0.6393

Epoch 30/30

 - 89s - loss: 0.6547 - accuracy: 0.6700 - val_loss: 0.6913 - val_accuracy: 0.6257

Test accuracy:

0.6257210969924927

--------------------------------------------------------------------------------

Model: "sequential_8"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 32)           5376

_________________________________________________________________

Dropout2_1 (Dropout)         (None, 128, 32)           0

_________________________________________________________________

LSTM2_2 (LSTM)               (None, 32)                8320

_________________________________________________________________
```

```
Dropout2_2 (Dropout)         (None, 32)                0

_________________________________________________________________

dense_8 (Dense)              (None, 6)                 198

=================================================================

Total params: 13,894

Trainable params: 13,894

Non-trainable params: 0

_________________________________________________________________

None

 27%|███████████                                | 4/15 [2:29:37<6:30:08, 2128.02s/trial, best loss:
-0.907024085521698]
```

```
C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: T
he `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 87s - loss: 1.4029 - accuracy: 0.3594 - val_loss: 1.3258 - val_accuracy: 0.3403

Epoch 2/30

 - 86s - loss: 1.5225 - accuracy: 0.3779 - val_loss: 1.2840 - val_accuracy: 0.3797

Epoch 3/30

 - 85s - loss: 1.1847 - accuracy: 0.4872 - val_loss: 1.1139 - val_accuracy: 0.4561

Epoch 4/30

 - 86s - loss: 0.9024 - accuracy: 0.5494 - val_loss: 1.0047 - val_accuracy: 0.5076

Epoch 5/30

 - 86s - loss: 0.8974 - accuracy: 0.5397 - val_loss: 0.8452 - val_accuracy: 0.5158

Epoch 6/30

 - 87s - loss: 0.8320 - accuracy: 0.5837 - val_loss: 0.8177 - val_accuracy: 0.6105

Epoch 7/30

 - 87s - loss: 0.8082 - accuracy: 0.6113 - val_loss: 0.8907 - val_accuracy: 0.5864

Epoch 8/30

 - 87s - loss: 0.9094 - accuracy: 0.5975 - val_loss: 1.6767 - val_accuracy: 0.2922

Epoch 9/30

 - 88s - loss: 1.1411 - accuracy: 0.5167 - val_loss: 1.2123 - val_accuracy: 0.5093

Epoch 10/30

 - 87s - loss: 0.9798 - accuracy: 0.5754 - val_loss: 1.0159 - val_accuracy: 0.5582
```

```
Epoch 11/30
 - 88s - loss: 0.9329 - accuracy: 0.5917 - val_loss: 1.0272 - val_accuracy: 0.5222
Epoch 12/30
 - 166s - loss: 0.9175 - accuracy: 0.5891 - val_loss: 0.9795 - val_accuracy: 0.5646
Epoch 13/30
 - 218s - loss: 0.8889 - accuracy: 0.6013 - val_loss: 0.9231 - val_accuracy: 0.5993
Epoch 14/30
 - 198s - loss: 0.8729 - accuracy: 0.6081 - val_loss: 1.0935 - val_accuracy: 0.5131
Epoch 15/30
 - 83s - loss: 0.8645 - accuracy: 0.6172 - val_loss: 1.0469 - val_accuracy: 0.4961
Epoch 16/30
 - 86s - loss: 0.8486 - accuracy: 0.6219 - val_loss: 0.9444 - val_accuracy: 0.5287
Epoch 17/30
 - 87s - loss: 0.8451 - accuracy: 0.6261 - val_loss: 1.0282 - val_accuracy: 0.4910
Epoch 18/30
 - 87s - loss: 0.8514 - accuracy: 0.6294 - val_loss: 0.9713 - val_accuracy: 0.4934
Epoch 19/30
 - 88s - loss: 0.8325 - accuracy: 0.6378 - val_loss: 0.9100 - val_accuracy: 0.5300
Epoch 20/30
 - 88s - loss: 0.8218 - accuracy: 0.6313 - val_loss: 1.0021 - val_accuracy: 0.5025
Epoch 21/30
 - 88s - loss: 0.8355 - accuracy: 0.6262 - val_loss: 1.0497 - val_accuracy: 0.4679
Epoch 22/30
 - 87s - loss: 0.8242 - accuracy: 0.6314 - val_loss: 1.0322 - val_accuracy: 0.4954
Epoch 23/30
 - 89s - loss: 0.8087 - accuracy: 0.6308 - val_loss: 0.9040 - val_accuracy: 0.5127
Epoch 24/30
 - 88s - loss: 0.8181 - accuracy: 0.6340 - val_loss: 0.9115 - val_accuracy: 0.5114
Epoch 25/30
 - 88s - loss: 0.8057 - accuracy: 0.6366 - val_loss: 0.8412 - val_accuracy: 0.5955
Epoch 26/30
```

```
Epoch 26/30

 - 89s - loss: 0.7921 - accuracy: 0.6436 - val_loss: 0.8887 - val_accuracy: 0.5351

Epoch 27/30

 - 88s - loss: 0.8292 - accuracy: 0.6308 - val_loss: 0.8700 - val_accuracy: 0.5813

Epoch 28/30

 - 88s - loss: 0.8302 - accuracy: 0.6302 - val_loss: 0.9249 - val_accuracy: 0.4903

Epoch 29/30

 - 89s - loss: 0.7972 - accuracy: 0.6325 - val_loss: 0.8830 - val_accuracy: 0.5110

Epoch 30/30

 - 89s - loss: 0.8060 - accuracy: 0.6394 - val_loss: 0.9190 - val_accuracy: 0.4985

Test accuracy:

0.49847301840782166

--------------------------------------------------------------------------------------

Model: "sequential_9"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 28)           4256

_________________________________________________________________

Dropout2_1 (Dropout)         (None, 128, 28)           0

_________________________________________________________________

LSTM2_2 (LSTM)               (None, 32)                7808

_________________________________________________________________

Dropout2_2 (Dropout)         (None, 32)                0

_________________________________________________________________

dense_9 (Dense)              (None, 6)                 198

=================================================================

Total params: 12,262

Trainable params: 12,262

Non-trainable params: 0

_________________________________________________________________

None

 33%|██████████████                             | 5/15 [3:18:38<6:35:19, 2371.91s/trial, best loss:
-0.907024085521698]

C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: T
he `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 89s - loss: 1.1065 - accuracy: 0.5162 - val_loss: 0.7742 - val_accuracy: 0.6142

Epoch 2/30

 - 88s - loss: 0.7655 - accuracy: 0.6435 - val_loss: 0.7721 - val_accuracy: 0.6159

Epoch 3/30

 - 89s - loss: 0.7252 - accuracy: 0.6498 - val_loss: 0.7953 - val_accuracy: 0.6206

Epoch 4/30

 - 88s - loss: 0.6873 - accuracy: 0.7228 - val_loss: 1.5207 - val_accuracy: 0.6115

Epoch 5/30

 - 89s - loss: 0.4735 - accuracy: 0.8283 - val_loss: 0.5483 - val_accuracy: 0.8775

Epoch 6/30

 - 89s - loss: 0.3874 - accuracy: 0.8985 - val_loss: 0.4441 - val_accuracy: 0.8870

Epoch 7/30

 - 88s - loss: 0.2733 - accuracy: 0.9240 - val_loss: 0.9576 - val_accuracy: 0.8602

Epoch 8/30

 - 88s - loss: 0.2396 - accuracy: 0.9290 - val_loss: 0.7027 - val_accuracy: 0.8904

Epoch 9/30

 - 88s - loss: 0.2612 - accuracy: 0.9336 - val_loss: 0.3995 - val_accuracy: 0.9233

Epoch 10/30

 - 88s - loss: 0.2104 - accuracy: 0.9416 - val_loss: 0.6544 - val_accuracy: 0.8955

Epoch 11/30

 - 88s - loss: 0.2307 - accuracy: 0.9368 - val_loss: 0.7530 - val_accuracy: 0.8955

Epoch 12/30

 - 88s - loss: 0.2608 - accuracy: 0.9385 - val_loss: 0.5262 - val_accuracy: 0.9070

Epoch 13/30

 - 87s - loss: 0.2032 - accuracy: 0.9418 - val_loss: 0.6210 - val_accuracy: 0.8962

Epoch 14/30

 - 86s - loss: 0.2921 - accuracy: 0.9312 - val_loss: 0.6547 - val_accuracy: 0.9196

Epoch 15/30

 - 85s - loss: 0.2165 - accuracy: 0.9397 - val_loss: 0.4533 - val_accuracy: 0.9043
```

```
Epoch 16/30
 - 85s - loss: 0.2341 - accuracy: 0.9378 - val_loss: 0.7403 - val_accuracy: 0.9036
Epoch 17/30
 - 90s - loss: 0.2237 - accuracy: 0.9415 - val_loss: 0.6848 - val_accuracy: 0.8985
Epoch 18/30
 - 89s - loss: 0.1760 - accuracy: 0.9450 - val_loss: 0.8258 - val_accuracy: 0.8839
Epoch 19/30
 - 84s - loss: 0.2151 - accuracy: 0.9434 - val_loss: 0.6551 - val_accuracy: 0.9033
Epoch 20/30
 - 85s - loss: 0.2099 - accuracy: 0.9450 - val_loss: 0.7713 - val_accuracy: 0.9053
Epoch 21/30
 - 84s - loss: 0.2723 - accuracy: 0.9422 - val_loss: 0.8540 - val_accuracy: 0.9006
Epoch 22/30
 - 84s - loss: 0.2461 - accuracy: 0.9415 - val_loss: 0.6597 - val_accuracy: 0.8975
Epoch 23/30
 - 84s - loss: 0.2432 - accuracy: 0.9421 - val_loss: 1.1084 - val_accuracy: 0.8694
Epoch 24/30
 - 85s - loss: 0.2483 - accuracy: 0.9460 - val_loss: 0.9005 - val_accuracy: 0.8870
Epoch 25/30
 - 85s - loss: 0.2246 - accuracy: 0.9455 - val_loss: 0.8369 - val_accuracy: 0.9006
Epoch 26/30
 - 92s - loss: 0.2236 - accuracy: 0.9459 - val_loss: 0.7147 - val_accuracy: 0.9070
Epoch 27/30
 - 89s - loss: 0.3482 - accuracy: 0.9328 - val_loss: 0.7364 - val_accuracy: 0.9114
Epoch 28/30
 - 90s - loss: 0.2553 - accuracy: 0.9449 - val_loss: 0.7258 - val_accuracy: 0.8972
Epoch 29/30
 - 88s - loss: 0.2523 - accuracy: 0.9450 - val_loss: 1.2224 - val_accuracy: 0.8548
Epoch 30/30
 - 89s - loss: 0.2592 - accuracy: 0.9437 - val_loss: 0.7607 - val_accuracy: 0.8850
Test accuracy:
```

```
0.8849677443504333

--------------------------------------------------------------------------------

Model: "sequential_10"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 38)           7296

_________________________________________________________________

Dropout2_1 (Dropout)         (None, 128, 38)           0

_________________________________________________________________

LSTM2_2 (LSTM)               (None, 32)                9088

_________________________________________________________________

Dropout2_2 (Dropout)         (None, 32)                0

_________________________________________________________________

dense_10 (Dense)             (None, 6)                 198

=================================================================

Total params: 16,582

Trainable params: 16,582

Non-trainable params: 0

_________________________________________________________________

None

 40%|████████████████                          | 6/15 [4:02:22<6:07:09, 2447.72s/trial, best loss: -0.907024085521698]
```

```
C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 92s - loss: 1.4028 - accuracy: 0.3693 - val_loss: 1.4806 - val_accuracy: 0.3268


Epoch 2/30

 - 90s - loss: 1.6998 - accuracy: 0.2450 - val_loss: 1.4646 - val_accuracy: 0.3549


Epoch 3/30

 - 94s - loss: 1.1518 - accuracy: 0.4268 - val_loss: 0.8213 - val_accuracy: 0.5280


Epoch 4/30

 - 92s - loss: 0.8811 - accuracy: 0.5326 - val_loss: 0.8785 - val_accuracy: 0.5161


Epoch 5/30

 - 87s - loss: 0.8235 - accuracy: 0.5471 - val_loss: 0.8026 - val_accuracy: 0.5948
```

```
Epoch 6/30

 - 87s - loss: 0.8074 - accuracy: 0.5707 - val_loss: 0.7690 - val_accuracy: 0.5341

Epoch 7/30

 - 93s - loss: 0.7816 - accuracy: 0.6114 - val_loss: 0.7227 - val_accuracy: 0.6020

Epoch 8/30

 - 89s - loss: 0.8134 - accuracy: 0.6177 - val_loss: 0.8257 - val_accuracy: 0.6060

Epoch 9/30

 - 89s - loss: 0.7151 - accuracy: 0.6510 - val_loss: 0.7714 - val_accuracy: 0.6091

Epoch 10/30

 - 87s - loss: 0.6783 - accuracy: 0.6628 - val_loss: 0.7171 - val_accuracy: 0.6305

Epoch 11/30

 - 86s - loss: 0.6861 - accuracy: 0.6617 - val_loss: 0.7313 - val_accuracy: 0.6315

Epoch 12/30

 - 86s - loss: 0.6655 - accuracy: 0.6636 - val_loss: 0.7326 - val_accuracy: 0.6295

Epoch 13/30

 - 85s - loss: 0.6630 - accuracy: 0.6610 - val_loss: 0.7263 - val_accuracy: 0.6247

Epoch 14/30

 - 84s - loss: 0.6508 - accuracy: 0.6691 - val_loss: 0.7529 - val_accuracy: 0.6281

Epoch 15/30

 - 87s - loss: 0.6783 - accuracy: 0.6642 - val_loss: 0.7431 - val_accuracy: 0.6332

Epoch 16/30

 - 85s - loss: 0.7175 - accuracy: 0.6595 - val_loss: 0.7363 - val_accuracy: 0.6274

Epoch 17/30

 - 86s - loss: 0.6859 - accuracy: 0.6608 - val_loss: 0.7326 - val_accuracy: 0.6312

Epoch 18/30

 - 88s - loss: 0.6408 - accuracy: 0.6714 - val_loss: 0.7136 - val_accuracy: 0.6332

Epoch 19/30

 - 87s - loss: 0.6383 - accuracy: 0.6669 - val_loss: 0.7105 - val_accuracy: 0.6481

Epoch 20/30

 - 90s - loss: 0.6389 - accuracy: 0.6653 - val_loss: 0.6866 - val_accuracy: 0.6305

Epoch 21/30
```

```
 - 97s - loss: 0.6172 - accuracy: 0.6763 - val_loss: 0.6783 - val_accuracy: 0.6356

Epoch 22/30

 - 104s - loss: 0.5832 - accuracy: 0.6717 - val_loss: 0.6646 - val_accuracy: 0.6457

Epoch 23/30

 - 114s - loss: 0.5713 - accuracy: 0.6863 - val_loss: 0.6654 - val_accuracy: 0.6576

Epoch 24/30

 - 89s - loss: 0.5417 - accuracy: 0.6918 - val_loss: 0.6497 - val_accuracy: 0.6362

Epoch 25/30

 - 98s - loss: 0.5354 - accuracy: 0.7042 - val_loss: 0.7415 - val_accuracy: 0.6305

Epoch 26/30

 - 94s - loss: 0.5877 - accuracy: 0.7187 - val_loss: 0.6414 - val_accuracy: 0.8093

Epoch 27/30

 - 93s - loss: 0.4003 - accuracy: 0.8553 - val_loss: 0.4129 - val_accuracy: 0.8704

Epoch 28/30

 - 95s - loss: 0.2786 - accuracy: 0.9215 - val_loss: 0.5572 - val_accuracy: 0.8351

Epoch 29/30

 - 89s - loss: 0.2497 - accuracy: 0.9320 - val_loss: 0.3488 - val_accuracy: 0.9094

Epoch 30/30

 - 106s - loss: 0.2549 - accuracy: 0.9294 - val_loss: 0.4230 - val_accuracy: 0.9036

Test accuracy:

0.903630793094635

-------------------------------------------------------------------------------------

Model: "sequential_11"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 32)           5376

_________________________________________________________________

Dropout2_1 (Dropout)         (None, 128, 32)           0

_________________________________________________________________

LSTM2_2 (LSTM)               (None, 26)                6136

_________________________________________________________________

Dropout2_2 (Dropout)         (None, 26)                0
```

```
_________________________________________________________________
dense_11 (Dense)             (None, 6)                 162
=================================================================
Total params: 11,674
Trainable params: 11,674
Non-trainable params: 0
_________________________________________________________________
None
 47%|████████████████████                      | 7/15 [4:48:12<5:38:26, 2538.32s/trial, best loss:
-0.907024085521698]
```

```
C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: T
he `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 89s - loss: 1.4429 - accuracy: 0.3845 - val_loss: 1.4236 - val_accuracy: 0.3644

Epoch 2/30
 - 88s - loss: 1.4552 - accuracy: 0.3568 - val_loss: 1.4463 - val_accuracy: 0.2952

Epoch 3/30
 - 87s - loss: 1.3964 - accuracy: 0.3770 - val_loss: 1.2669 - val_accuracy: 0.4075

Epoch 4/30
 - 87s - loss: 1.3693 - accuracy: 0.4100 - val_loss: 1.3018 - val_accuracy: 0.4483

Epoch 5/30
 - 86s - loss: 1.4013 - accuracy: 0.3901 - val_loss: 1.5134 - val_accuracy: 0.3271

Epoch 6/30
 - 87s - loss: 1.3426 - accuracy: 0.4036 - val_loss: 1.3231 - val_accuracy: 0.4092

Epoch 7/30
 - 94s - loss: 1.4005 - accuracy: 0.3953 - val_loss: 1.3728 - val_accuracy: 0.3604

Epoch 8/30
 - 85s - loss: 1.0156 - accuracy: 0.5209 - val_loss: 0.9976 - val_accuracy: 0.5093

Epoch 9/30
 - 87s - loss: 0.8941 - accuracy: 0.5726 - val_loss: 0.8686 - val_accuracy: 0.5755

Epoch 10/30
 - 84s - loss: 0.8395 - accuracy: 0.6054 - val_loss: 0.8718 - val_accuracy: 0.5887

Epoch 11/30
```

```
 - 85s - loss: 0.8336 - accuracy: 0.6178 - val_loss: 1.1002 - val_accuracy: 0.4238

Epoch 12/30

 - 84s - loss: 0.8639 - accuracy: 0.5941 - val_loss: 0.9172 - val_accuracy: 0.5687

Epoch 13/30

 - 84s - loss: 0.7897 - accuracy: 0.6408 - val_loss: 0.8316 - val_accuracy: 0.6305

Epoch 14/30

 - 91s - loss: 0.7312 - accuracy: 0.6508 - val_loss: 1.2319 - val_accuracy: 0.4703

Epoch 15/30

 - 85s - loss: 0.6585 - accuracy: 0.6712 - val_loss: 0.6782 - val_accuracy: 0.6189

Epoch 16/30

 - 85s - loss: 0.6590 - accuracy: 0.7065 - val_loss: 0.8245 - val_accuracy: 0.6121

Epoch 17/30

 - 84s - loss: 0.6067 - accuracy: 0.7545 - val_loss: 0.5028 - val_accuracy: 0.7496

Epoch 18/30

 - 84s - loss: 0.4805 - accuracy: 0.7792 - val_loss: 0.5102 - val_accuracy: 0.7469

Epoch 19/30

 - 85s - loss: 0.5280 - accuracy: 0.7723 - val_loss: 0.6928 - val_accuracy: 0.7299

Epoch 20/30

 - 85s - loss: 0.4971 - accuracy: 0.7946 - val_loss: 0.6112 - val_accuracy: 0.7489

Epoch 21/30

 - 85s - loss: 0.4371 - accuracy: 0.8030 - val_loss: 0.6623 - val_accuracy: 0.7424

Epoch 22/30

 - 84s - loss: 0.4148 - accuracy: 0.8277 - val_loss: 0.6895 - val_accuracy: 0.7102

Epoch 23/30

 - 85s - loss: 0.3827 - accuracy: 0.8683 - val_loss: 0.6784 - val_accuracy: 0.8273

Epoch 24/30

 - 85s - loss: 0.3235 - accuracy: 0.9082 - val_loss: 0.5518 - val_accuracy: 0.8721

Epoch 25/30

 - 85s - loss: 0.3771 - accuracy: 0.8972 - val_loss: 0.5631 - val_accuracy: 0.8731

Epoch 26/30

 - 84s - loss: 0.3171 - accuracy: 0.9165 - val_loss: 0.5193 - val_accuracy: 0.8850
```

```
Epoch 27/30

 - 85s - loss: 0.2784 - accuracy: 0.9282 - val_loss: 0.4890 - val_accuracy: 0.8894

Epoch 28/30

 - 85s - loss: 0.2519 - accuracy: 0.9329 - val_loss: 0.5299 - val_accuracy: 0.8670

Epoch 29/30

 - 85s - loss: 0.3011 - accuracy: 0.9221 - val_loss: 0.5365 - val_accuracy: 0.8616

Epoch 30/30

 - 85s - loss: 0.3052 - accuracy: 0.9230 - val_loss: 0.4292 - val_accuracy: 0.8972

Test accuracy:

0.8971835970878601

-------------------------------------------------------------------------------------

Model: "sequential_12"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM1_1 (LSTM)               (None, 28)                4256

_________________________________________________________________

Dropout1_1 (Dropout)         (None, 28)                0

_________________________________________________________________

dense_12 (Dense)             (None, 6)                 174

=================================================================

Total params: 4,430

Trainable params: 4,430

Non-trainable params: 0

_________________________________________________________________

None

 53%|███████████████████████▏                    | 8/15 [5:31:10<4:57:31, 2550.24s/trial, best loss: -0.907024085521698]
```

```
C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 42s - loss: 1.1995 - accuracy: 0.4825 - val_loss: 0.8156 - val_accuracy: 0.6040

Epoch 2/30

 - 41s - loss: 0.7401 - accuracy: 0.6519 - val_loss: 0.7209 - val_accuracy: 0.6597
```

```
Epoch 3/30
 - 42s - loss: 0.6116 - accuracy: 0.7689 - val_loss: 0.6222 - val_accuracy: 0.7777
Epoch 4/30
 - 42s - loss: 0.4831 - accuracy: 0.8417 - val_loss: 0.5554 - val_accuracy: 0.8646
Epoch 5/30
 - 42s - loss: 0.3303 - accuracy: 0.9105 - val_loss: 0.6170 - val_accuracy: 0.8687
Epoch 6/30
 - 41s - loss: 0.2725 - accuracy: 0.9253 - val_loss: 0.3685 - val_accuracy: 0.8778
Epoch 7/30
 - 42s - loss: 0.2708 - accuracy: 0.9290 - val_loss: 0.3785 - val_accuracy: 0.9016
Epoch 8/30
 - 42s - loss: 0.2427 - accuracy: 0.9350 - val_loss: 0.5175 - val_accuracy: 0.8795
Epoch 9/30
 - 42s - loss: 0.2336 - accuracy: 0.9348 - val_loss: 0.4530 - val_accuracy: 0.9019
Epoch 10/30
 - 43s - loss: 0.2182 - accuracy: 0.9366 - val_loss: 0.5972 - val_accuracy: 0.8894
Epoch 11/30
 - 41s - loss: 0.2085 - accuracy: 0.9411 - val_loss: 0.4844 - val_accuracy: 0.8823
Epoch 12/30
 - 42s - loss: 0.1940 - accuracy: 0.9402 - val_loss: 0.4508 - val_accuracy: 0.8744
Epoch 13/30
 - 42s - loss: 0.1977 - accuracy: 0.9410 - val_loss: 0.4119 - val_accuracy: 0.8918
Epoch 14/30
 - 41s - loss: 0.1798 - accuracy: 0.9414 - val_loss: 0.4438 - val_accuracy: 0.8985
Epoch 15/30
 - 41s - loss: 0.1954 - accuracy: 0.9400 - val_loss: 0.4258 - val_accuracy: 0.8985
Epoch 16/30
 - 42s - loss: 0.1872 - accuracy: 0.9442 - val_loss: 0.4605 - val_accuracy: 0.9104
Epoch 17/30
 - 42s - loss: 0.1857 - accuracy: 0.9425 - val_loss: 0.3549 - val_accuracy: 0.9111
Epoch 18/30
```

```
 - 41s - loss: 0.1738 - accuracy: 0.9445 - val_loss: 0.5261 - val_accuracy: 0.8884

Epoch 19/30

 - 42s - loss: 0.1692 - accuracy: 0.9453 - val_loss: 0.3933 - val_accuracy: 0.9158

Epoch 20/30

 - 42s - loss: 0.1675 - accuracy: 0.9468 - val_loss: 0.4342 - val_accuracy: 0.8955

Epoch 21/30

 - 41s - loss: 0.1595 - accuracy: 0.9505 - val_loss: 0.3704 - val_accuracy: 0.9104

Epoch 22/30

 - 42s - loss: 0.1763 - accuracy: 0.9448 - val_loss: 0.4188 - val_accuracy: 0.9074

Epoch 23/30

 - 41s - loss: 0.1688 - accuracy: 0.9499 - val_loss: 0.3751 - val_accuracy: 0.9043

Epoch 24/30

 - 42s - loss: 0.1580 - accuracy: 0.9478 - val_loss: 0.4637 - val_accuracy: 0.9006

Epoch 25/30

 - 41s - loss: 0.1642 - accuracy: 0.9453 - val_loss: 0.6123 - val_accuracy: 0.9009

Epoch 26/30

 - 42s - loss: 0.1615 - accuracy: 0.9505 - val_loss: 0.6108 - val_accuracy: 0.8816

Epoch 27/30

 - 42s - loss: 0.1754 - accuracy: 0.9482 - val_loss: 0.5656 - val_accuracy: 0.9009

Epoch 28/30

 - 41s - loss: 0.1565 - accuracy: 0.9498 - val_loss: 0.5331 - val_accuracy: 0.8951

Epoch 29/30

 - 41s - loss: 0.1633 - accuracy: 0.9486 - val_loss: 0.4717 - val_accuracy: 0.8907

Epoch 30/30

 - 42s - loss: 0.1744 - accuracy: 0.9460 - val_loss: 0.5124 - val_accuracy: 0.9070

Test accuracy:

0.907024085521698

-----------------------------------------------------------------------------------

Model: "sequential_13"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================
```

```
LSTM1_1 (LSTM)              (None, 28)                4256

_________________________________________________________________

Dropout1_1 (Dropout)        (None, 28)                0

_________________________________________________________________

dense_13 (Dense)            (None, 6)                 174

=================================================================

Total params: 4,430

Trainable params: 4,430

Non-trainable params: 0

_________________________________________________________________

None

Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 42s - loss: 1.0027 - accuracy: 0.5328 - val_loss: 0.9019 - val_accuracy: 0.5317

Epoch 2/30

 - 42s - loss: 0.7269 - accuracy: 0.6621 - val_loss: 0.6872 - val_accuracy: 0.7163

Epoch 3/30

 - 41s - loss: 0.4744 - accuracy: 0.8267 - val_loss: 0.5662 - val_accuracy: 0.8544

Epoch 4/30

 - 42s - loss: 0.3070 - accuracy: 0.9125 - val_loss: 0.6067 - val_accuracy: 0.8609

Epoch 5/30

 - 41s - loss: 0.2665 - accuracy: 0.9249 - val_loss: 0.6176 - val_accuracy: 0.8588

Epoch 6/30

 - 42s - loss: 0.2549 - accuracy: 0.9301 - val_loss: 0.5581 - val_accuracy: 0.8802

Epoch 7/30

 - 41s - loss: 0.2479 - accuracy: 0.9328 - val_loss: 0.6646 - val_accuracy: 0.8877

Epoch 8/30

 - 42s - loss: 0.2436 - accuracy: 0.9339 - val_loss: 0.7024 - val_accuracy: 0.8809

Epoch 9/30

 - 42s - loss: 0.1977 - accuracy: 0.9355 - val_loss: 0.7326 - val_accuracy: 0.8907

Epoch 10/30

 - 42s - loss: 0.2233 - accuracy: 0.9357 - val_loss: 0.6647 - val_accuracy: 0.8534

Epoch 11/30
```

```
 - 42s - loss: 0.2169 - accuracy: 0.9402 - val_loss: 1.0692 - val_accuracy: 0.8704

Epoch 12/30

 - 42s - loss: 0.2106 - accuracy: 0.9408 - val_loss: 0.5532 - val_accuracy: 0.8941

Epoch 13/30

 - 42s - loss: 0.1861 - accuracy: 0.9419 - val_loss: 0.7593 - val_accuracy: 0.8968

Epoch 14/30

 - 42s - loss: 0.2113 - accuracy: 0.9411 - val_loss: 0.7086 - val_accuracy: 0.8795

Epoch 15/30

 - 41s - loss: 0.1960 - accuracy: 0.9419 - val_loss: 0.5328 - val_accuracy: 0.8846

Epoch 16/30

 - 41s - loss: 0.1966 - accuracy: 0.9423 - val_loss: 0.5590 - val_accuracy: 0.8890

Epoch 17/30

 - 42s - loss: 0.2425 - accuracy: 0.9403 - val_loss: 0.8190 - val_accuracy: 0.8799

Epoch 18/30

 - 41s - loss: 0.1772 - accuracy: 0.9461 - val_loss: 0.8568 - val_accuracy: 0.8887

Epoch 19/30

 - 41s - loss: 0.2090 - accuracy: 0.9429 - val_loss: 0.7266 - val_accuracy: 0.8894

Epoch 20/30

 - 41s - loss: 0.1993 - accuracy: 0.9459 - val_loss: 0.8223 - val_accuracy: 0.8768

Epoch 21/30

 - 41s - loss: 0.2151 - accuracy: 0.9438 - val_loss: 0.6573 - val_accuracy: 0.8938

Epoch 22/30

 - 42s - loss: 0.2015 - accuracy: 0.9450 - val_loss: 0.6055 - val_accuracy: 0.9101

Epoch 23/30

 - 42s - loss: 0.2298 - accuracy: 0.9421 - val_loss: 0.7030 - val_accuracy: 0.8972

Epoch 24/30

 - 42s - loss: 0.2003 - accuracy: 0.9429 - val_loss: 0.7307 - val_accuracy: 0.8816

Epoch 25/30

 - 42s - loss: 0.2100 - accuracy: 0.9427 - val_loss: 0.5635 - val_accuracy: 0.8958

Epoch 26/30

 - 42s - loss: 0.1723 - accuracy: 0.9493 - val_loss: 0.8766 - val_accuracy: 0.8602
```

```
Epoch 27/30

 - 42s - loss: 0.1906 - accuracy: 0.9460 - val_loss: 0.6078 - val_accuracy: 0.8968

Epoch 28/30

 - 42s - loss: 0.1715 - accuracy: 0.9463 - val_loss: 0.7454 - val_accuracy: 0.8823

Epoch 29/30

 - 42s - loss: 0.1809 - accuracy: 0.9449 - val_loss: 0.6643 - val_accuracy: 0.8921

Epoch 30/30

 - 41s - loss: 0.1725 - accuracy: 0.9448 - val_loss: 0.9032 - val_accuracy: 0.8785

Test accuracy:

0.8785205483436584

--------------------------------------------------------------------------------------

Model: "sequential_14"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM1_1 (LSTM)               (None, 32)                5376

_________________________________________________________________

Dropout1_1 (Dropout)         (None, 32)                0

_________________________________________________________________

dense_14 (Dense)             (None, 6)                 198

=================================================================

Total params: 5,574

Trainable params: 5,574

Non-trainable params: 0

_________________________________________________________________

None

Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 42s - loss: 1.0644 - accuracy: 0.5354 - val_loss: 0.8381 - val_accuracy: 0.5962

Epoch 2/30

 - 42s - loss: 0.6650 - accuracy: 0.7292 - val_loss: 0.7218 - val_accuracy: 0.7401

Epoch 3/30

 - 42s - loss: 0.4828 - accuracy: 0.8523 - val_loss: 0.6350 - val_accuracy: 0.8334

Epoch 4/30

 - 41s - loss: 0.3176 - accuracy: 0.9162 - val_loss: 0.7805 - val_accuracy: 0.8134
```

```
 - 41s - loss: 0.3176 - accuracy: 0.9162 - val_loss: 0.7805 - val_accuracy: 0.8134

Epoch 5/30

 - 42s - loss: 0.2737 - accuracy: 0.9246 - val_loss: 0.3831 - val_accuracy: 0.8924

Epoch 6/30

 - 42s - loss: 0.2679 - accuracy: 0.9298 - val_loss: 0.3104 - val_accuracy: 0.9094

Epoch 7/30

 - 42s - loss: 0.2250 - accuracy: 0.9351 - val_loss: 0.3892 - val_accuracy: 0.9057

Epoch 8/30

 - 41s - loss: 0.2185 - accuracy: 0.9376 - val_loss: 0.3251 - val_accuracy: 0.9111

Epoch 9/30

 - 41s - loss: 0.2044 - accuracy: 0.9408 - val_loss: 0.3463 - val_accuracy: 0.9019

Epoch 10/30

 - 41s - loss: 0.1974 - accuracy: 0.9374 - val_loss: 0.3243 - val_accuracy: 0.9172

Epoch 11/30

 - 41s - loss: 0.1959 - accuracy: 0.9411 - val_loss: 0.4819 - val_accuracy: 0.8873

Epoch 12/30

 - 41s - loss: 0.1873 - accuracy: 0.9410 - val_loss: 0.2982 - val_accuracy: 0.9080

Epoch 13/30

 - 40s - loss: 0.1886 - accuracy: 0.9410 - val_loss: 0.4888 - val_accuracy: 0.8921

Epoch 14/30

 - 41s - loss: 0.1792 - accuracy: 0.9431 - val_loss: 0.5286 - val_accuracy: 0.8941

Epoch 15/30

 - 41s - loss: 0.1787 - accuracy: 0.9446 - val_loss: 1.7401 - val_accuracy: 0.8195

Epoch 16/30

 - 41s - loss: 0.1737 - accuracy: 0.9422 - val_loss: 0.4108 - val_accuracy: 0.8999

Epoch 17/30

 - 41s - loss: 0.1815 - accuracy: 0.9483 - val_loss: 0.3457 - val_accuracy: 0.9169

Epoch 18/30

 - 41s - loss: 0.1782 - accuracy: 0.9445 - val_loss: 0.4221 - val_accuracy: 0.8985

Epoch 19/30

 - 41s - loss: 0.1702 - accuracy: 0.9464 - val_loss: 1.1094 - val_accuracy: 0.8429
```

```
Epoch 20/30
 - 41s - loss: 0.1536 - accuracy: 0.9494 - val_loss: 0.3667 - val_accuracy: 0.9016
Epoch 21/30
 - 41s - loss: 0.1587 - accuracy: 0.9460 - val_loss: 0.5853 - val_accuracy: 0.8795
Epoch 22/30
 - 40s - loss: 0.1665 - accuracy: 0.9445 - val_loss: 0.3583 - val_accuracy: 0.9094
Epoch 23/30
 - 41s - loss: 0.1700 - accuracy: 0.9444 - val_loss: 0.4412 - val_accuracy: 0.9009
Epoch 24/30
 - 41s - loss: 0.1514 - accuracy: 0.9486 - val_loss: 0.8159 - val_accuracy: 0.8704
Epoch 25/30
 - 40s - loss: 0.1552 - accuracy: 0.9504 - val_loss: 0.4666 - val_accuracy: 0.8744
Epoch 26/30
 - 41s - loss: 0.1587 - accuracy: 0.9455 - val_loss: 0.3381 - val_accuracy: 0.9080
Epoch 27/30
 - 41s - loss: 0.1593 - accuracy: 0.9502 - val_loss: 0.3881 - val_accuracy: 0.9209
Epoch 28/30
 - 41s - loss: 0.1480 - accuracy: 0.9498 - val_loss: 0.5789 - val_accuracy: 0.8778
Epoch 29/30
 - 42s - loss: 0.1466 - accuracy: 0.9472 - val_loss: 0.4506 - val_accuracy: 0.9019
Epoch 30/30
 - 42s - loss: 0.1555 - accuracy: 0.9489 - val_loss: 1.1520 - val_accuracy: 0.8493
Test accuracy:
0.8493382930755615
---------------------------------------------------------------------------------------
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
LSTM1_1 (LSTM)               (None, 36)                6624
_________________________________________________________________
Dropout1_1 (Dropout)         (None, 36)                0
_________________________________________________________________
```

```
dense_15 (Dense)             (None, 6)                 222

=================================================================

Total params: 6,846

Trainable params: 6,846

Non-trainable params: 0

_________________________________________________________________

None

Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 42s - loss: 1.7172 - accuracy: 0.3772 - val_loss: 2.6505 - val_accuracy: 0.1815

Epoch 2/30

 - 42s - loss: 2.3384 - accuracy: 0.1916 - val_loss: 1.9345 - val_accuracy: 0.1822

Epoch 3/30

 - 41s - loss: 1.6603 - accuracy: 0.2980 - val_loss: 1.2615 - val_accuracy: 0.5959

Epoch 4/30

 - 42s - loss: 0.9577 - accuracy: 0.6007 - val_loss: 0.9756 - val_accuracy: 0.6050

Epoch 5/30

 - 42s - loss: 0.8506 - accuracy: 0.6363 - val_loss: 0.8093 - val_accuracy: 0.6162

Epoch 6/30

 - 42s - loss: 0.7400 - accuracy: 0.6541 - val_loss: 0.7589 - val_accuracy: 0.6223

Epoch 7/30

 - 42s - loss: 0.6750 - accuracy: 0.6600 - val_loss: 0.7013 - val_accuracy: 0.6288

Epoch 8/30

 - 42s - loss: 0.6864 - accuracy: 0.6700 - val_loss: 0.6981 - val_accuracy: 0.6244

Epoch 9/30

 - 42s - loss: 0.5954 - accuracy: 0.7442 - val_loss: 0.5732 - val_accuracy: 0.7859

Epoch 10/30

 - 42s - loss: 0.4576 - accuracy: 0.8568 - val_loss: 0.5138 - val_accuracy: 0.8690

Epoch 11/30

 - 42s - loss: 0.3627 - accuracy: 0.9083 - val_loss: 0.4326 - val_accuracy: 0.8911

Epoch 12/30

 - 42s - loss: 0.3568 - accuracy: 0.9104 - val_loss: 0.8080 - val_accuracy: 0.8354

Epoch 13/30
```

```
Epoch 13/30

 - 42s - loss: 0.2915 - accuracy: 0.9297 - val_loss: 0.3880 - val_accuracy: 0.8928

Epoch 14/30

 - 42s - loss: 0.2546 - accuracy: 0.9351 - val_loss: 0.4385 - val_accuracy: 0.8758

Epoch 15/30

 - 42s - loss: 0.2443 - accuracy: 0.9321 - val_loss: 0.3946 - val_accuracy: 0.8863

Epoch 16/30

 - 42s - loss: 0.3717 - accuracy: 0.9174 - val_loss: 0.5241 - val_accuracy: 0.8931

Epoch 17/30

 - 42s - loss: 0.3524 - accuracy: 0.9188 - val_loss: 0.3976 - val_accuracy: 0.8982

Epoch 18/30

 - 41s - loss: 0.2434 - accuracy: 0.9348 - val_loss: 0.4799 - val_accuracy: 0.8948

Epoch 19/30

 - 42s - loss: 0.2341 - accuracy: 0.9334 - val_loss: 0.8313 - val_accuracy: 0.8354

Epoch 20/30

 - 42s - loss: 0.2273 - accuracy: 0.9377 - val_loss: 0.3591 - val_accuracy: 0.9002

Epoch 21/30

 - 42s - loss: 0.2353 - accuracy: 0.9391 - val_loss: 0.5165 - val_accuracy: 0.8958

Epoch 22/30

 - 42s - loss: 0.2023 - accuracy: 0.9418 - val_loss: 0.3882 - val_accuracy: 0.9087

Epoch 23/30

 - 41s - loss: 0.2138 - accuracy: 0.9378 - val_loss: 0.5192 - val_accuracy: 0.8765

Epoch 24/30

 - 42s - loss: 0.2364 - accuracy: 0.9346 - val_loss: 0.4609 - val_accuracy: 0.8982

Epoch 25/30

 - 42s - loss: 0.1848 - accuracy: 0.9431 - val_loss: 0.4760 - val_accuracy: 0.8955

Epoch 26/30

 - 42s - loss: 0.1692 - accuracy: 0.9460 - val_loss: 0.4407 - val_accuracy: 0.8948

Epoch 27/30

 - 42s - loss: 0.2013 - accuracy: 0.9456 - val_loss: 1.2388 - val_accuracy: 0.7957

Epoch 28/30
```

```
 - 42s - loss: 0.2407 - accuracy: 0.9373 - val_loss: 0.5032 - val_accuracy: 0.8863

Epoch 29/30

 - 42s - loss: 0.1731 - accuracy: 0.9471 - val_loss: 0.4254 - val_accuracy: 0.8975

Epoch 30/30

 - 42s - loss: 0.1834 - accuracy: 0.9472 - val_loss: 0.4322 - val_accuracy: 0.9057

Test accuracy:

0.9056667685508728

--------------------------------------------------------------------------------------

Model: "sequential_16"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 38)           7296

_________________________________________________________________

Dropout2_1 (Dropout)         (None, 128, 38)           0

_________________________________________________________________

LSTM2_2 (LSTM)               (None, 32)                9088

_________________________________________________________________

Dropout2_2 (Dropout)         (None, 32)                0

_________________________________________________________________

dense_16 (Dense)             (None, 6)                 198

=================================================================

Total params: 16,582

Trainable params: 16,582

Non-trainable params: 0

_________________________________________________________________

None

Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 85s - loss: 1.2902 - accuracy: 0.4484 - val_loss: 1.5059 - val_accuracy: 0.4523

Epoch 2/30

 - 85s - loss: 0.8470 - accuracy: 0.6119 - val_loss: 0.7527 - val_accuracy: 0.6186

Epoch 3/30

 - 84s - loss: 0.7618 - accuracy: 0.6325 - val_loss: 0.7686 - val_accuracy: 0.5996

Epoch 4/30

 - 85s - loss: 0.7249 - accuracy: 0.6411 - val_loss: 0.7385 - val_accuracy: 0.6291
```

```
Epoch 5/30

 - 84s - loss: 0.7052 - accuracy: 0.6401 - val_loss: 0.7687 - val_accuracy: 0.6247

Epoch 6/30

 - 85s - loss: 0.6954 - accuracy: 0.6748 - val_loss: 0.6727 - val_accuracy: 0.7411

Epoch 7/30

 - 83s - loss: 0.5304 - accuracy: 0.8112 - val_loss: 0.5850 - val_accuracy: 0.8378

Epoch 8/30

 - 83s - loss: 0.3669 - accuracy: 0.9059 - val_loss: 0.6429 - val_accuracy: 0.8415

Epoch 9/30

 - 84s - loss: 0.3178 - accuracy: 0.9104 - val_loss: 0.5785 - val_accuracy: 0.8660

Epoch 10/30

 - 84s - loss: 0.2716 - accuracy: 0.9283 - val_loss: 0.5399 - val_accuracy: 0.8873

Epoch 11/30

 - 84s - loss: 0.2844 - accuracy: 0.9237 - val_loss: 0.4108 - val_accuracy: 0.8979

Epoch 12/30

 - 84s - loss: 0.2427 - accuracy: 0.9298 - val_loss: 0.3965 - val_accuracy: 0.8938

Epoch 13/30

 - 84s - loss: 0.2359 - accuracy: 0.9297 - val_loss: 0.4198 - val_accuracy: 0.8955

Epoch 14/30

 - 83s - loss: 0.2198 - accuracy: 0.9332 - val_loss: 0.7411 - val_accuracy: 0.8799

Epoch 15/30

 - 84s - loss: 0.2301 - accuracy: 0.9369 - val_loss: 0.4898 - val_accuracy: 0.8935

Epoch 16/30

 - 85s - loss: 0.2255 - accuracy: 0.9293 - val_loss: 0.3784 - val_accuracy: 0.9125

Epoch 17/30

 - 84s - loss: 0.2197 - accuracy: 0.9363 - val_loss: 0.4776 - val_accuracy: 0.8890

Epoch 18/30

 - 84s - loss: 0.2219 - accuracy: 0.9344 - val_loss: 0.4574 - val_accuracy: 0.8918

Epoch 19/30

 - 84s - loss: 0.2097 - accuracy: 0.9378 - val_loss: 0.5725 - val_accuracy: 0.8880
```

```
Epoch 20/30

 - 84s - loss: 0.2226 - accuracy: 0.9369 - val_loss: 0.5674 - val_accuracy: 0.8846

Epoch 21/30

 - 84s - loss: 0.1974 - accuracy: 0.9374 - val_loss: 0.4346 - val_accuracy: 0.9084

Epoch 22/30

 - 84s - loss: 0.2044 - accuracy: 0.9400 - val_loss: 0.4241 - val_accuracy: 0.8962

Epoch 23/30

 - 84s - loss: 0.2053 - accuracy: 0.9382 - val_loss: 0.3971 - val_accuracy: 0.9264

Epoch 24/30

 - 84s - loss: 0.1805 - accuracy: 0.9415 - val_loss: 0.5722 - val_accuracy: 0.8985

Epoch 25/30

 - 84s - loss: 0.1880 - accuracy: 0.9419 - val_loss: 0.4023 - val_accuracy: 0.9087

Epoch 26/30

 - 84s - loss: 0.1916 - accuracy: 0.9446 - val_loss: 0.4804 - val_accuracy: 0.9125

Epoch 27/30

 - 84s - loss: 0.1814 - accuracy: 0.9457 - val_loss: 0.5394 - val_accuracy: 0.8928

Epoch 28/30

 - 84s - loss: 0.1812 - accuracy: 0.9449 - val_loss: 0.4782 - val_accuracy: 0.9091

Epoch 29/30

 - 84s - loss: 0.1889 - accuracy: 0.9434 - val_loss: 0.4688 - val_accuracy: 0.9094

Epoch 30/30

 - 84s - loss: 0.1727 - accuracy: 0.9448 - val_loss: 0.5210 - val_accuracy: 0.9108

Test accuracy:

0.9107567071914673

-------------------------------------------------------------------------------------

Model: "sequential_17"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM1_1 (LSTM)               (None, 32)                5376

_________________________________________________________________

Dropout1_1 (Dropout)         (None, 32)                0

_________________________________________________________________
```

```
dense_17 (Dense)             (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
None
 87%|██████████████████████████████████▊     | 13/15 [7:36:28<1:01:42, 1851.42s/trial, best loss: -0.9107567071914673]
```

```
C:\Users\sesha\Untitled Folder\7. Human Activity Recognition\HAR\temp_model.py:338: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
 - 42s - loss: 1.2385 - accuracy: 0.4886 - val_loss: 1.0258 - val_accuracy: 0.5097

Epoch 2/30
 - 42s - loss: 0.8457 - accuracy: 0.6155 - val_loss: 0.8643 - val_accuracy: 0.6451

Epoch 3/30
 - 42s - loss: 0.7264 - accuracy: 0.6741 - val_loss: 0.8262 - val_accuracy: 0.6342

Epoch 4/30
 - 42s - loss: 0.7200 - accuracy: 0.7790 - val_loss: 0.7531 - val_accuracy: 0.7635

Epoch 5/30
 - 41s - loss: 0.5599 - accuracy: 0.8428 - val_loss: 0.7138 - val_accuracy: 0.8124

Epoch 6/30
 - 42s - loss: 0.4144 - accuracy: 0.9048 - val_loss: 0.4870 - val_accuracy: 0.8877

Epoch 7/30
 - 41s - loss: 0.3175 - accuracy: 0.9251 - val_loss: 0.4806 - val_accuracy: 0.8853

Epoch 8/30
 - 41s - loss: 0.4273 - accuracy: 0.9121 - val_loss: 0.5479 - val_accuracy: 0.8918

Epoch 9/30
 - 42s - loss: 0.3306 - accuracy: 0.9285 - val_loss: 0.4977 - val_accuracy: 0.8911

Epoch 10/30
 - 42s - loss: 0.3662 - accuracy: 0.9123 - val_loss: 0.4246 - val_accuracy: 0.8894

Epoch 11/30
 - 41s - loss: 0.2709 - accuracy: 0.9319 - val_loss: 0.4307 - val_accuracy: 0.8901
```

```
Epoch 12/30
 - 42s - loss: 0.2634 - accuracy: 0.9376 - val_loss: 0.5554 - val_accuracy: 0.8731
Epoch 13/30
 - 41s - loss: 0.2314 - accuracy: 0.9415 - val_loss: 0.3742 - val_accuracy: 0.8982
Epoch 14/30
 - 42s - loss: 0.2037 - accuracy: 0.9418 - val_loss: 0.3445 - val_accuracy: 0.8914
Epoch 15/30
 - 41s - loss: 0.2091 - accuracy: 0.9421 - val_loss: 0.3776 - val_accuracy: 0.9070
Epoch 16/30
 - 42s - loss: 0.1759 - accuracy: 0.9448 - val_loss: 0.3679 - val_accuracy: 0.8999
Epoch 17/30
 - 42s - loss: 0.1772 - accuracy: 0.9437 - val_loss: 0.4783 - val_accuracy: 0.8856
Epoch 18/30
 - 42s - loss: 0.1833 - accuracy: 0.9448 - val_loss: 0.4555 - val_accuracy: 0.8985
Epoch 19/30
 - 41s - loss: 0.2109 - accuracy: 0.9415 - val_loss: 0.3878 - val_accuracy: 0.8996
Epoch 20/30
 - 42s - loss: 0.1806 - accuracy: 0.9464 - val_loss: 0.3337 - val_accuracy: 0.9043
Epoch 21/30
 - 42s - loss: 0.1826 - accuracy: 0.9436 - val_loss: 0.4490 - val_accuracy: 0.8975
Epoch 22/30
 - 42s - loss: 0.1667 - accuracy: 0.9459 - val_loss: 0.4339 - val_accuracy: 0.8792
Epoch 23/30
 - 42s - loss: 0.2107 - accuracy: 0.9442 - val_loss: 0.4799 - val_accuracy: 0.8914
Epoch 24/30
 - 41s - loss: 0.1716 - accuracy: 0.9482 - val_loss: 0.3420 - val_accuracy: 0.9114
Epoch 25/30
 - 41s - loss: 0.1671 - accuracy: 0.9472 - val_loss: 0.4044 - val_accuracy: 0.8951
Epoch 26/30
 - 41s - loss: 0.1758 - accuracy: 0.9460 - val_loss: 0.4542 - val_accuracy: 0.8989
```

```
Epoch 27/30

 - 41s - loss: 0.1610 - accuracy: 0.9486 - val_loss: 0.3684 - val_accuracy: 0.9104

Epoch 28/30

 - 41s - loss: 0.1630 - accuracy: 0.9471 - val_loss: 0.3572 - val_accuracy: 0.9053

Epoch 29/30

 - 41s - loss: 0.1870 - accuracy: 0.9421 - val_loss: 0.4220 - val_accuracy: 0.8945

Epoch 30/30

 - 41s - loss: 0.1859 - accuracy: 0.9440 - val_loss: 0.4118 - val_accuracy: 0.9125

Test accuracy:

0.9124533534049988

---------------------------------------------------------------------------------------

Model: "sequential_18"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

LSTM2_1 (LSTM)               (None, 128, 32)           5376

_________________________________________________________________

Dropout2_1 (Dropout)         (None, 128, 32)           0

_________________________________________________________________

LSTM2_2 (LSTM)               (None, 32)                8320

_________________________________________________________________

Dropout2_2 (Dropout)         (None, 32)                0

_________________________________________________________________

dense_18 (Dense)             (None, 6)                 198

=================================================================

Total params: 13,894

Trainable params: 13,894

Non-trainable params: 0

_________________________________________________________________

None

Train on 7352 samples, validate on 2947 samples

Epoch 1/30

 - 83s - loss: 1.4513 - accuracy: 0.3604 - val_loss: 1.3985 - val_accuracy: 0.3536

Epoch 2/30

 - 83s - loss: 1.3158 - accuracy: 0.3972 - val_loss: 1.0826 - val_accuracy: 0.4520

Epoch 3/30
```

```
Epoch 3/30

 - 82s - loss: 1.6688 - accuracy: 0.2977 - val_loss: 1.4572 - val_accuracy: 0.3492

Epoch 4/30

 - 84s - loss: 1.4494 - accuracy: 0.3587 - val_loss: 1.2590 - val_accuracy: 0.5297

Epoch 5/30

 - 84s - loss: 0.9886 - accuracy: 0.4986 - val_loss: 0.9313 - val_accuracy: 0.5053

Epoch 6/30

 - 84s - loss: 0.8593 - accuracy: 0.5207 - val_loss: 0.9031 - val_accuracy: 0.5219

Epoch 7/30

 - 84s - loss: 0.9414 - accuracy: 0.5143 - val_loss: 0.8160 - val_accuracy: 0.5310

Epoch 8/30

 - 84s - loss: 0.7982 - accuracy: 0.5442 - val_loss: 0.8071 - val_accuracy: 0.5290

Epoch 9/30

 - 84s - loss: 0.8534 - accuracy: 0.5329 - val_loss: 0.8109 - val_accuracy: 0.5304

Epoch 10/30

 - 85s - loss: 0.8339 - accuracy: 0.5343 - val_loss: 0.8371 - val_accuracy: 0.5300

Epoch 11/30

 - 85s - loss: 0.8657 - accuracy: 0.5174 - val_loss: 0.9562 - val_accuracy: 0.5304

Epoch 12/30

 - 84s - loss: 0.8601 - accuracy: 0.5381 - val_loss: 0.7980 - val_accuracy: 0.5310

Epoch 13/30

 - 87s - loss: 1.0228 - accuracy: 0.4773 - val_loss: 0.8270 - val_accuracy: 0.5310

Epoch 14/30

 - 85s - loss: 0.8018 - accuracy: 0.5424 - val_loss: 0.8031 - val_accuracy: 0.5304

Epoch 15/30

 - 84s - loss: 0.8179 - accuracy: 0.5379 - val_loss: 0.8376 - val_accuracy: 0.5280

Epoch 16/30

 - 84s - loss: 0.7949 - accuracy: 0.5427 - val_loss: 0.8129 - val_accuracy: 0.5300

Epoch 17/30

 - 84s - loss: 0.7943 - accuracy: 0.5514 - val_loss: 0.8684 - val_accuracy: 0.5239

Epoch 18/30

 - 84s - loss: 0.8029 - accuracy: 0.5405 - val_loss: 0.8750 - val_accuracy: 0.5212
```

```
Epoch 19/30

 - 85s - loss: 0.7960 - accuracy: 0.5343 - val_loss: 0.7871 - val_accuracy: 0.5307

Epoch 20/30

 - 84s - loss: 0.7808 - accuracy: 0.5407 - val_loss: 0.7772 - val_accuracy: 0.5307

Epoch 21/30

 - 84s - loss: 0.7694 - accuracy: 0.5447 - val_loss: 0.7761 - val_accuracy: 0.5314

Epoch 22/30

 - 84s - loss: 0.7850 - accuracy: 0.5420 - val_loss: 0.7979 - val_accuracy: 0.5253

Epoch 23/30

 - 84s - loss: 0.7763 - accuracy: 0.5409 - val_loss: 0.7911 - val_accuracy: 0.5365

Epoch 24/30

 - 84s - loss: 0.8573 - accuracy: 0.5427 - val_loss: 0.8198 - val_accuracy: 0.5277

Epoch 25/30

 - 84s - loss: 0.7933 - accuracy: 0.5973 - val_loss: 0.7459 - val_accuracy: 0.5948

Epoch 26/30

 - 85s - loss: 0.7079 - accuracy: 0.6401 - val_loss: 0.7293 - val_accuracy: 0.6295

Epoch 27/30

 - 85s - loss: 0.7159 - accuracy: 0.6402 - val_loss: 0.7108 - val_accuracy: 0.6335

Epoch 28/30

 - 84s - loss: 0.6783 - accuracy: 0.6567 - val_loss: 0.7112 - val_accuracy: 0.6305

Epoch 29/30

 - 83s - loss: 0.6678 - accuracy: 0.6560 - val_loss: 0.7568 - val_accuracy: 0.6189

Epoch 30/30

 - 85s - loss: 0.6527 - accuracy: 0.6620 - val_loss: 0.7085 - val_accuracy: 0.6284

Test accuracy:

0.6284356713294983

-----------------------------------------------------------------------------------

100%|██████████████████████████████████████████| 15/15 [8:39:24<00:00, 2077.64s/trial, best loss:
-0.9124533534049988]
```

In [52]:

```python
total_trials = dict()
for t, trial in enumerate(trials):
    vals = trial.get('misc').get('vals')
```

```python
        vals = trial.get('misc').get('vals')
        print('Model',t+1,'parameters')
        print(vals)
        print()
        z = eval_hyperopt_space(space, vals)
        total_trials['M'+str(t+1)] = z
        print(z)
        print('------------------------------------------------')
```

```
Model 1 parameters
{'Dropout': [0.36598023572757926], 'Dropout_1': [0.6047146037530785], 'Dropout_2':
[0.5188826519950874], 'LSTM': [0], 'LSTM_1': [1], 'LSTM_2': [1], 'choiceval': [1], 'if': [0],
'l2': [0.00016900597529479822], 'l2_1': [0.0006108763092812357], 'l2_2': [0.0007371698374615214],
'lr': [0.01942874904782045], 'lr_1': [0.015993860150909475]}

{'Dropout': 0.36598023572757926, 'Dropout_1': 0.6047146037530785, 'Dropout_2': 0.5188826519950874,
'LSTM': 28, 'LSTM_1': 32, 'LSTM_2': 32, 'choiceval': 'rmsprop', 'if': 'one', 'l2':
0.00016900597529479822, 'l2_1': 0.0006108763092812357, 'l2_2': 0.0007371698374615214, 'lr':
0.01942874904782045, 'lr_1': 0.015993860150909475}
------------------------------------------------
Model 2 parameters
{'Dropout': [0.604072168386432], 'Dropout_1': [0.5642077861572957], 'Dropout_2':
[0.4689742513688654], 'LSTM': [0], 'LSTM_1': [1], 'LSTM_2': [0], 'choiceval': [1], 'if': [1],
'l2': [2.221286943616341e-06], 'l2_1': [0.0009770005173795487], 'l2_2': [0.0008366666847115819], '
lr': [0.023605271151689124], 'lr_1': [0.015140941766877332]}

{'Dropout': 0.604072168386432, 'Dropout_1': 0.5642077861572957, 'Dropout_2': 0.4689742513688654, '
LSTM': 28, 'LSTM_1': 32, 'LSTM_2': 28, 'choiceval': 'rmsprop', 'if': 'two', 'l2':
2.221286943616341e-06, 'l2_1': 0.0009770005173795487, 'l2_2': 0.0008366666847115819, 'lr':
0.023605271151689124, 'lr_1': 0.015140941766877332}
------------------------------------------------
Model 3 parameters
{'Dropout': [0.649118836907314], 'Dropout_1': [0.6408661828169875], 'Dropout_2':
[0.5025116318997556], 'LSTM': [2], 'LSTM_1': [2], 'LSTM_2': [1], 'choiceval': [1], 'if': [1],
'l2': [0.00011247630115130428], 'l2_1': [0.0003949936266626689], 'l2_2': [0.0009758185183456943],
'lr': [0.013618600574440736], 'lr_1': [0.014402022095061829]}

{'Dropout': 0.649118836907314, 'Dropout_1': 0.6408661828169875, 'Dropout_2': 0.5025116318997556, '
LSTM': 38, 'LSTM_1': 36, 'LSTM_2': 32, 'choiceval': 'rmsprop', 'if': 'two', 'l2':
0.00011247630115130428, 'l2_1': 0.0003949936266626689, 'l2_2': 0.0009758185183456943, 'lr':
0.013618600574440736, 'lr_1': 0.014402022095061829}
------------------------------------------------
Model 4 parameters
{'Dropout': [0.5709919477993022], 'Dropout_1': [0.6574295784428639], 'Dropout_2':
[0.39377498664819843], 'LSTM': [1], 'LSTM_1': [1], 'LSTM_2': [2], 'choiceval': [0], 'if': [1], 'l2
': [0.00019824027740992625], 'l2_1': [0.0007646166765488501], 'l2_2': [0.00041266207281071243], 'l
r': [0.01675112837971219], 'lr_1': [0.009417276849790152]}

{'Dropout': 0.5709919477993022, 'Dropout_1': 0.6574295784428639, 'Dropout_2': 0.39377498664819843,
'LSTM': 32, 'LSTM_1': 32, 'LSTM_2': 36, 'choiceval': 'adam', 'if': 'two', 'l2':
0.00019824027740992625, 'l2_1': 0.0007646166765488501, 'l2_2': 0.00041266207281071243, 'lr':
0.01675112837971219, 'lr_1': 0.009417276849790152}
------------------------------------------------
Model 5 parameters
{'Dropout': [0.48051787644406624], 'Dropout_1': [0.5744163772727372], 'Dropout_2':
[0.5086629864785656], 'LSTM': [1], 'LSTM_1': [1], 'LSTM_2': [0], 'choiceval': [0], 'if': [1],
'l2': [2.749908849077252e-05], 'l2_1': [0.000587606728324542], 'l2_2': [0.0003746350041674067], 'l
r': [0.01834130504525777], 'lr_1': [0.0229410270349058]}

{'Dropout': 0.48051787644406624, 'Dropout_1': 0.5744163772727372, 'Dropout_2': 0.5086629864785656,
'LSTM': 32, 'LSTM_1': 32, 'LSTM_2': 28, 'choiceval': 'adam', 'if': 'two', 'l2':
2.749908849077252e-05, 'l2_1': 0.000587606728324542, 'l2_2': 0.0003746350041674067, 'lr':
0.01834130504525777, 'lr_1': 0.0229410270349058}
------------------------------------------------
Model 6 parameters
{'Dropout': [0.5813560517914963], 'Dropout_1': [0.6046109124722276], 'Dropout_2':
[0.5355832635290444], 'LSTM': [0], 'LSTM_1': [1], 'LSTM_2': [2], 'choiceval': [1], 'if': [1],
'l2': [1.612769130873457e-05], 'l2_1': [0.0009772817488940724], 'l2_2': [0.0006883693507416478], '
lr': [0.017446396677831936], 'lr_1': [0.015805655140931824]}

{'Dropout': 0.5813560517914963, 'Dropout_1': 0.6046109124722276, 'Dropout_2': 0.5355832635290444,
'LSTM': 28, 'LSTM_1': 32, 'LSTM_2': 36, 'choiceval': 'rmsprop', 'if': 'two', 'l2':
1.612769130873457e-05, 'l2_1': 0.0009772817488940724, 'l2_2': 0.0006883693507416478, 'lr':
0.017446396677831936, 'lr_1': 0.015805655140931824}
------------------------------------------------
Model 7 parameters
{'Dropout': [0.5293597400197904], 'Dropout_1': [0.5958807193410454], 'Dropout_2':
```

```
[0.42617520692074906], 'LSTM': [2], 'LSTM_1': [1], 'LSTM_2': [2], 'choiceval': [0], 'if': [1], 'l2
': [4.567626225804864e-05], 'l2_1': [0.0005422412690636627], 'l2_2': [0.00033351393608141357], 'lr
': [0.01068491666284852], 'lr_1': [0.01643494651558678]}

{'Dropout': 0.5293597400197904, 'Dropout_1': 0.5958807193410454, 'Dropout_2': 0.42617520692074906,
'LSTM': 38, 'LSTM_1': 32, 'LSTM_2': 36, 'choiceval': 'adam', 'if': 'two', 'l2':
4.567626225804864e-05, 'l2_1': 0.0005422412690636627, 'l2_2': 0.00033351393608141357, 'lr':
0.01068491666284852, 'lr_1': 0.01643494651558678}
-------------------------------------------------
Model 8 parameters
{'Dropout': [0.5950749367948185], 'Dropout_1': [0.5997621117444732], 'Dropout_2':
[0.4999621572265873], 'LSTM': [1], 'LSTM_1': [0], 'LSTM_2': [1], 'choiceval': [0], 'if': [1],
'l2': [5.865420439323175e-05], 'l2_1': [0.0007302305870589934], 'l2_2': [0.000258985915829989], 'l
r': [0.010314137826059229], 'lr_1': [0.009310543992889801]}

{'Dropout': 0.5950749367948185, 'Dropout_1': 0.5997621117444732, 'Dropout_2': 0.4999621572265873,
'LSTM': 32, 'LSTM_1': 26, 'LSTM_2': 32, 'choiceval': 'adam', 'if': 'two', 'l2':
5.865420439323175e-05, 'l2_1': 0.0007302305870589934, 'l2_2': 0.000258985915829989, 'lr':
0.010314137826059229, 'lr_1': 0.009310543992889801}
-------------------------------------------------
Model 9 parameters
{'Dropout': [0.45037579382108217], 'Dropout_1': [0.6781762554752515], 'Dropout_2':
[0.4794831735512747], 'LSTM': [1], 'LSTM_1': [1], 'LSTM_2': [0], 'choiceval': [1], 'if': [0],
'l2': [5.201497156118029e-05], 'l2_1': [0.0006257491042113806], 'l2_2': [0.0004437546321946204], '
lr': [0.023536039320918772], 'lr_1': [0.012611516495429879]}

{'Dropout': 0.45037579382108217, 'Dropout_1': 0.6781762554752515, 'Dropout_2': 0.4794831735512747,
'LSTM': 32, 'LSTM_1': 32, 'LSTM_2': 28, 'choiceval': 'rmsprop', 'if': 'one', 'l2':
5.201497156118029e-05, 'l2_1': 0.0006257491042113806, 'l2_2': 0.0004437546321946204, 'lr':
0.023536039320918772, 'lr_1': 0.012611516495429879}
-------------------------------------------------
Model 10 parameters
{'Dropout': [0.45714950357785966], 'Dropout_1': [0.6894085538291769], 'Dropout_2':
[0.45216713875784914], 'LSTM': [0], 'LSTM_1': [1], 'LSTM_2': [0], 'choiceval': [1], 'if': [0], 'l2
': [7.681307310729229e-05], 'l2_1': [0.0004143619965361732], 'l2_2': [9.225974322037534e-05], 'lr'
: [0.01235075833910319], 'lr_1': [0.018058999803996133]}

{'Dropout': 0.45714950357785966, 'Dropout_1': 0.6894085538291769, 'Dropout_2':
0.45216713875784914, 'LSTM': 28, 'LSTM_1': 32, 'LSTM_2': 28, 'choiceval': 'rmsprop', 'if': 'one',
'l2': 7.681307310729229e-05, 'l2_1': 0.0004143619965361732, 'l2_2': 9.225974322037534e-05, 'lr': 0
.01235075833910319, 'lr_1': 0.018058999803996133}
-------------------------------------------------
Model 11 parameters
{'Dropout': [0.5808002757682877], 'Dropout_1': [0.660514929179723], 'Dropout_2':
[0.4733734305745834], 'LSTM': [1], 'LSTM_1': [0], 'LSTM_2': [1], 'choiceval': [1], 'if': [0],
'l2': [0.0001195365208222095], 'l2_1': [0.0001849314123467004], 'l2_2': [0.0005106207029550342], '
lr': [0.013696392786995321], 'lr_1': [0.009420957669947726]}

{'Dropout': 0.5808002757682877, 'Dropout_1': 0.660514929179723, 'Dropout_2': 0.4733734305745834, '
LSTM': 32, 'LSTM_1': 26, 'LSTM_2': 32, 'choiceval': 'rmsprop', 'if': 'one', 'l2':
0.0001195365208222095, 'l2_1': 0.0001849314123467004, 'l2_2': 0.0005106207029550342, 'lr':
0.013696392786995321, 'lr_1': 0.009420957669947726}
-------------------------------------------------
Model 12 parameters
{'Dropout': [0.5666044972741778], 'Dropout_1': [0.5837804766498599], 'Dropout_2':
[0.38708976069745693], 'LSTM': [1], 'LSTM_1': [2], 'LSTM_2': [2], 'choiceval': [0], 'if': [0], 'l2
': [6.379888690521487e-05], 'l2_1': [0.00013256157391301627], 'l2_2': [0.0009457487322332761], 'lr
': [0.021003723896153827], 'lr_1': [0.014111778261744532]}

{'Dropout': 0.5666044972741778, 'Dropout_1': 0.5837804766498599, 'Dropout_2': 0.38708976069745693,
'LSTM': 32, 'LSTM_1': 36, 'LSTM_2': 36, 'choiceval': 'adam', 'if': 'one', 'l2':
6.379888690521487e-05, 'l2_1': 0.00013256157391301627, 'l2_2': 0.0009457487322332761, 'lr':
0.021003723896153827, 'lr_1': 0.014111778261744532}
-------------------------------------------------
Model 13 parameters
{'Dropout': [0.47945603666694214], 'Dropout_1': [0.6410658485741121], 'Dropout_2':
[0.431428962525653], 'LSTM': [2], 'LSTM_1': [1], 'LSTM_2': [2], 'choiceval': [1], 'if': [1], 'l2':
[0.00018573736431464218], 'l2_1': [0.0009992918522039433], 'l2_2': [0.000376241262719619], 'lr': [
0.02028522715636994], 'lr_1': [0.02075108210315991]}

{'Dropout': 0.47945603666694214, 'Dropout_1': 0.6410658485741121, 'Dropout_2': 0.431428962525653,
'LSTM': 38, 'LSTM_1': 32, 'LSTM_2': 36, 'choiceval': 'rmsprop', 'if': 'two', 'l2':
0.00018573736431464218, 'l2_1': 0.0009992918522039433, 'l2_2': 0.000376241262719619, 'lr':
0.02028522715636994, 'lr_1': 0.02075108210315991}
-------------------------------------------------
Model 14 parameters
{'Dropout': [0.3802031741395868], 'Dropout_1': [0.6903389204823146], 'Dropout_2':
```

```
[0.3654341425327902], 'LSTM': [2], 'LSTM_1': [2], 'LSTM_2': [1], 'choiceval': [0], 'if': [0], 
'l2': [0.00015208023802140732], 'l2_1': [0.000643128044948208], 'l2_2': [0.0007102309264917989], '
lr': [0.016347608866364167], 'lr_1': [0.024543333891182614]}

{'Dropout': 0.3802031741395868, 'Dropout_1': 0.6903389204823146, 'Dropout_2': 0.3654341425327902, 
'LSTM': 38, 'LSTM_1': 36, 'LSTM_2': 32, 'choiceval': 'adam', 'if': 'one', 'l2': 
0.00015208023802140732, 'l2_1': 0.000643128044948208, 'l2_2': 0.0007102309264917989, 'lr': 
0.016347608866364167, 'lr_1': 0.024543333891182614}
------------------------------------------------
Model 15 parameters
{'Dropout': [0.578227610775208], 'Dropout_1': [0.6959943282933752], 'Dropout_2': 
[0.4519332465495095], 'LSTM': [1], 'LSTM_1': [1], 'LSTM_2': [1], 'choiceval': [0], 'if': [1], 
'l2': [9.909767403125834e-05], 'l2_1': [0.0004671776323322324], 'l2_2': [0.0008869747685138522], '
lr': [0.010099240007717829], 'lr_1': [0.024293576282946767]}

{'Dropout': 0.578227610775208, 'Dropout_1': 0.6959943282933752, 'Dropout_2': 0.4519332465495095, '
LSTM': 32, 'LSTM_1': 32, 'LSTM_2': 32, 'choiceval': 'adam', 'if': 'two', 'l2': 9.909767403125834e-
05, 'l2_1': 0.0004671776323322324, 'l2_2': 0.0008869747685138522, 'lr': 0.010099240007717829, 
'lr_1': 0.024293576282946767}
------------------------------------------------
```

In [53]:

```
best_run
```

Out[53]:

```
{'Dropout': 0.3802031741395868,
 'Dropout_1': 0.6903389204823146,
 'Dropout_2': 0.3654341425327902,
 'LSTM': 2,
 'LSTM_1': 2,
 'LSTM_2': 1,
 'choiceval': 0,
 'if': 0,
 'l2': 0.00015208023802140732,
 'l2_1': 0.000643128044948208,
 'l2_2': 0.0007102309264917989,
 'lr': 0.016347608866364167,
 'lr_1': 0.024543333891182614}
```

In [54]:

```
#BEST MODEL PARAMS
total_trials['M14']
```

Out[54]:

```
{'Dropout': 0.3802031741395868,
 'Dropout_1': 0.6903389204823146,
 'Dropout_2': 0.3654341425327902,
 'LSTM': 38,
 'LSTM_1': 36,
 'LSTM_2': 32,
 'choiceval': 'adam',
 'if': 'one',
 'l2': 0.00015208023802140732,
 'l2_1': 0.000643128044948208,
 'l2_2': 0.0007102309264917989,
 'lr': 0.016347608866364167,
 'lr_1': 0.024543333891182614}
```

In [55]:

```
#layes of best model
best_model.layers
```

Out[55]:

```
[<keras.layers.recurrent.LSTM at 0x202311cddc8>,
 <keras.layers.core.Dropout at 0x202311e0208>,
 <keras.layers.core.Dense at 0x202347c9408>]
```

In [56]:

```
X_train, Y_train, X_val, Y_val = data()
```

In [57]:

```
_,val_acc = best_model.evaluate(X_val, Y_val, verbose=0)
_,train_acc = best_model.evaluate(X_train, Y_train, verbose=0)
print('Train_accuracy',train_acc)
print('validation accuracy',val_acc)
```

```
Train_accuracy 0.9518498182296753
validation accuracy 0.9124533534049988
```

In [58]:

```
import sklearn.metrics as metrics
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Utility function to print the confusion matrix
def confusion_matrix_rnn(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    #return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
    return metrics.confusion_matrix(Y_true, Y_pred)
```

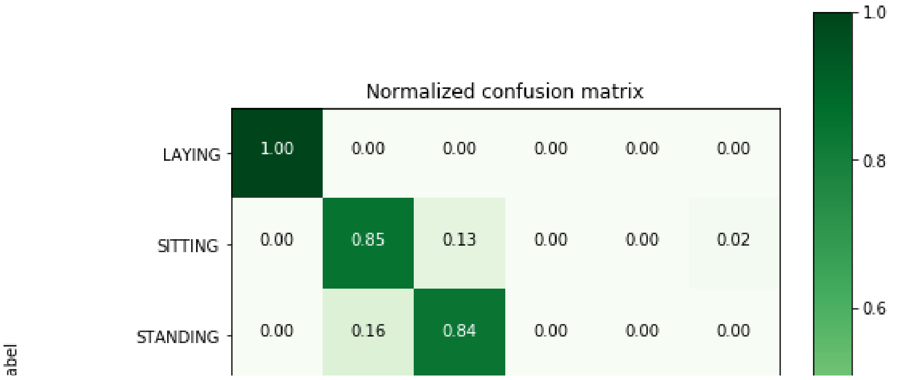
In [59]:

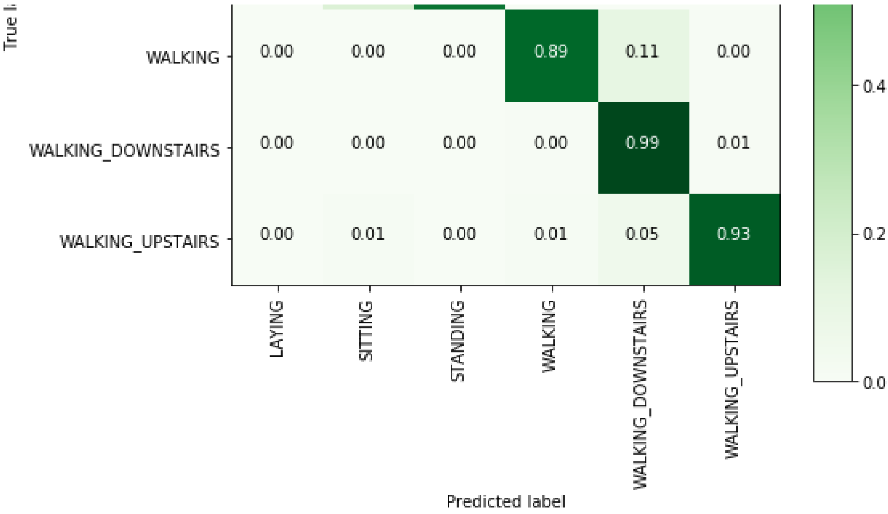
```
# Confusion Matrix
print(confusion_matrix_rnn(Y_val, best_model.predict(X_val)))
```

```
[[537   0   0   0   0   0]
 [  0 416  63   0   0  12]
 [  0  86 445   0   0   1]
 [  0   1   0 440  53   2]
 [  0   0   0   2 415   3]
 [  0   4   0   6  25 436]]
```

In [60]:

```
plt.figure(figsize=(8,8))
cm = confusion_matrix_rnn(Y_val, best_model.predict(X_val))
plot_confusion_matrix(cm, classes=labels, normalize=True, title='Normalized confusion matrix',
cmap = plt.cm.Greens)
plt.show()
```





1) There are misclassifications for Standing, Sitting, Walking and Walking_Upstairs.

2) Laying and Walking_downstairs have been classified perfectly.

3) a)Train Accuracy = ~96%

b)Validation Accuracy = ~92%

8.2) Convolutional Neural Network

In [61]:

```
import os
os.environ['PYTHONHASHSEED'] = '0'
import random as rn
np.random.seed(36)
rn.seed(36)
#tf.set_random_seed(36)
tf.random.set_seed(36)
# Force TensorFlow to use single thread.
# Multiple threads are a potential source of non-reproducible results.
# For further details, see: https://stackoverflow.com/questions/42022950/
session_conf = tf.compat.v1.ConfigProto(intra_op_parallelism_threads=1,
                              inter_op_parallelism_threads=1)

#tf.set_random_seed(36)
tf.random.set_seed(36)

sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [62]:

```
X_train, Y_train, X_val, Y_val = data()
```

In [63]:

```
###Scling data
from sklearn.preprocessing import StandardScaler
from sklearn.base import BaseEstimator, TransformerMixin
class scaling_tseries_data(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.scale = None

    def transform(self, X):
        temp_X1 = X.reshape((X.shape[0] * X.shape[1], X.shape[2]))
        temp_X1 = self.scale.transform(temp_X1)
        return temp_X1.reshape(X.shape)

    def fit(self, X):
        # remove overlaping
        remove = int(X.shape[1] / 2)
```

```
        temp_X = X[:, -remove:, :]
        # flatten data
        temp_X = temp_X.reshape((temp_X.shape[0] * temp_X.shape[1], temp_X.shape[2]))
        scale = StandardScaler()
        scale.fit(temp_X)
        self.scale = scale
        return self
```

In [64]:

```
Scale = scaling_tseries_data()
Scale.fit(X_train)
X_train_sc = Scale.transform(X_train)
X_val_sc = Scale.transform(X_val)
```

In [65]:

```
print('Shape of scaled X train',X_train_sc.shape)
print('Shape of scaled X test',X_val_sc.shape)
```

```
Shape of scaled X train (7352, 128, 9)
Shape of scaled X test (2947, 128, 9)
```

8.2.1 With basic number of Conv Layers

In [66]:

```
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical
from keras.layers import Flatten
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer='he_uniform',input
_shape=(128,9)))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer='he_uniform'))
model.add(Dropout(0.6))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(6, activation='softmax'))
model.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 126, 32)           896
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 124, 32)           3104
_________________________________________________________________
dropout_6 (Dropout)          (None, 124, 32)           0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 62, 32)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1984)              0
_________________________________________________________________
dense_19 (Dense)             (None, 50)                99250
_________________________________________________________________
dense_20 (Dense)             (None, 6)                 306
=================================================================
Total params: 103,556
Trainable params: 103,556
Non-trainable params: 0
_________________________________________________________________
```

In [67]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [68]:

```
model.fit(X_train_sc,Y_train, epochs=30, batch_size=16,validation_data=(X_val_sc, Y_val), verbose=1
)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 6s 775us/step - loss: 0.4866 - accuracy: 0.8109 - val
_loss: 0.3663 - val_accuracy: 0.8772
Epoch 2/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.1449 - accuracy: 0.9411 - val
_loss: 0.2930 - val_accuracy: 0.9050
Epoch 3/30
7352/7352 [==============================] - 3s 371us/step - loss: 0.1125 - accuracy: 0.9516 - val
_loss: 0.2247 - val_accuracy: 0.9118
Epoch 4/30
7352/7352 [==============================] - 3s 376us/step - loss: 0.1178 - accuracy: 0.9521 - val
_loss: 0.2168 - val_accuracy: 0.9158
Epoch 5/30
7352/7352 [==============================] - 3s 367us/step - loss: 0.0931 - accuracy: 0.9593 - val
_loss: 0.2341 - val_accuracy: 0.9230
Epoch 6/30
7352/7352 [==============================] - 3s 375us/step - loss: 0.0833 - accuracy: 0.9634 - val
_loss: 0.2361 - val_accuracy: 0.9277
Epoch 7/30
7352/7352 [==============================] - 3s 367us/step - loss: 0.0826 - accuracy: 0.9659 - val
_loss: 0.3160 - val_accuracy: 0.9074
Epoch 8/30
7352/7352 [==============================] - 3s 375us/step - loss: 0.0742 - accuracy: 0.9667 - val
_loss: 0.2566 - val_accuracy: 0.9203
Epoch 9/30
7352/7352 [==============================] - 3s 369us/step - loss: 0.0777 - accuracy: 0.9668 - val
_loss: 0.2314 - val_accuracy: 0.9325
Epoch 10/30
7352/7352 [==============================] - 3s 384us/step - loss: 0.0713 - accuracy: 0.9661 - val
_loss: 0.2545 - val_accuracy: 0.9186
Epoch 11/30
7352/7352 [==============================] - 3s 385us/step - loss: 0.0676 - accuracy: 0.9697 - val
_loss: 0.2696 - val_accuracy: 0.9281
Epoch 12/30
7352/7352 [==============================] - 3s 382us/step - loss: 0.0599 - accuracy: 0.9736 - val
_loss: 0.2541 - val_accuracy: 0.9230
Epoch 13/30
7352/7352 [==============================] - 3s 377us/step - loss: 0.0520 - accuracy: 0.9769 - val
_loss: 0.2797 - val_accuracy: 0.9264
Epoch 14/30
7352/7352 [==============================] - 3s 384us/step - loss: 0.0583 - accuracy: 0.9744 - val
_loss: 0.2935 - val_accuracy: 0.9230
Epoch 15/30
7352/7352 [==============================] - 3s 376us/step - loss: 0.0697 - accuracy: 0.9746 - val
_loss: 0.3338 - val_accuracy: 0.9186
Epoch 16/30
7352/7352 [==============================] - 3s 386us/step - loss: 0.0487 - accuracy: 0.9784 - val
_loss: 0.3396 - val_accuracy: 0.9257
Epoch 17/30
7352/7352 [==============================] - 3s 375us/step - loss: 0.0492 - accuracy: 0.9769 - val
_loss: 0.3285 - val_accuracy: 0.9104
Epoch 18/30
7352/7352 [==============================] - 3s 379us/step - loss: 0.0520 - accuracy: 0.9773 - val
_loss: 0.3402 - val_accuracy: 0.9257
Epoch 19/30
7352/7352 [==============================] - 3s 373us/step - loss: 0.0468 - accuracy: 0.9795 - val
_loss: 0.3623 - val_accuracy: 0.9270
Epoch 20/30
7352/7352 [==============================] - 3s 372us/step - loss: 0.0470 - accuracy: 0.9805 - val
_loss: 0.3856 - val_accuracy: 0.9192
Epoch 21/30
7352/7352 [==============================] - 3s 370us/step - loss: 0.0632 - accuracy: 0.9784 - val
_loss: 0.4059 - val_accuracy: 0.9175
Epoch 22/30
7352/7352 [==============================] - 3s 375us/step - loss: 0.0531 - accuracy: 0.9792 - val
_loss: 0.3688 - val_accuracy: 0.9270
Epoch 23/30
7352/7352 [==============================] - 3s 380us/step - loss: 0.0472 - accuracy: 0.9814 - val
_loss: 0.3779 - val_accuracy: 0.9141
Epoch 24/30
7352/7352 [==============================] - 3s 384us/step - loss: 0.0415 - accuracy: 0.9822 - val
```

```
7352/7352 [==============================] - 3s 384us/step - loss: 0.0415 - accuracy: 0.9822 - val
_loss: 0.3960 - val_accuracy: 0.9318
Epoch 25/30
7352/7352 [==============================] - 3s 378us/step - loss: 0.0351 - accuracy: 0.9845 - val
_loss: 0.3525 - val_accuracy: 0.9233
Epoch 26/30
7352/7352 [==============================] - 3s 387us/step - loss: 0.0362 - accuracy: 0.9853 - val
_loss: 0.4000 - val_accuracy: 0.9237
Epoch 27/30
7352/7352 [==============================] - 3s 371us/step - loss: 0.0449 - accuracy: 0.9812 - val
_loss: 0.4569 - val_accuracy: 0.9169
Epoch 28/30
7352/7352 [==============================] - 3s 379us/step - loss: 0.0332 - accuracy: 0.9850 - val
_loss: 0.3565 - val_accuracy: 0.9240
Epoch 29/30
7352/7352 [==============================] - 3s 385us/step - loss: 0.0315 - accuracy: 0.9859 - val
_loss: 0.4361 - val_accuracy: 0.9335
Epoch 30/30
7352/7352 [==============================] - 3s 373us/step - loss: 0.0303 - accuracy: 0.9864 - val
_loss: 0.4972 - val_accuracy: 0.9287
```

Out[68]:

```
<keras.callbacks.callbacks.History at 0x2021ab69ac8>
```

The train loss and Validation loss has a lotof difference, hence there are high chances that this is overfitting.

So we add Regularization.

In [69]:

```python
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',kernel_initializer='he_uniform',
                 kernel_regularizer=l2(0.1),input_shape=(128,9)))
model.add(Conv1D(filters=16, kernel_size=3, activation='relu',kernel_regularizer=l2(0.06),kernel_in
itializer='he_uniform'))
model.add(Dropout(0.65))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(6, activation='softmax'))
model.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_3 (Conv1D)            (None, 126, 32)           896
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 124, 16)           1552
_________________________________________________________________
dropout_7 (Dropout)          (None, 124, 16)           0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 62, 16)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 992)               0
_________________________________________________________________
dense_21 (Dense)             (None, 32)                31776
_________________________________________________________________
dense_22 (Dense)             (None, 6)                 198
=================================================================
Total params: 34,422
Trainable params: 34,422
Non-trainable params: 0
_________________________________________________________________
```

In [70]:

```python
import math
adam = keras.optimizers.Adam(lr=0.001)
rmsprop = keras.optimizers.RMSprop(lr=0.001)
def step_decay(epoch):
    return float(0.001 * math.pow(0.6, math.floor((1+epoch)/10)))
from keras.callbacks import LearningRateScheduler
```

```
lrate = LearningRateScheduler(step_decay)
callbacks_list = [lrate]

model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
```

In [71]:

```
model.fit(X_train_sc,Y_train, epochs=30, batch_size=16,validation_data=(X_val_sc, Y_val), verbose=1
)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 3s 422us/step - loss: 3.9381 - accuracy: 0.7990 - val
_loss: 1.4057 - val_accuracy: 0.8751
Epoch 2/30
7352/7352 [==============================] - 3s 413us/step - loss: 0.6835 - accuracy: 0.9215 - val
_loss: 0.6216 - val_accuracy: 0.8799
Epoch 3/30
7352/7352 [==============================] - 3s 390us/step - loss: 0.3596 - accuracy: 0.9302 - val
_loss: 0.5702 - val_accuracy: 0.8459
Epoch 4/30
7352/7352 [==============================] - 3s 376us/step - loss: 0.2930 - accuracy: 0.9320 - val
_loss: 0.4818 - val_accuracy: 0.8765
Epoch 5/30
7352/7352 [==============================] - 3s 379us/step - loss: 0.2618 - accuracy: 0.9368 - val
_loss: 0.4129 - val_accuracy: 0.8826
Epoch 6/30
7352/7352 [==============================] - 3s 379us/step - loss: 0.2561 - accuracy: 0.9347 - val
_loss: 0.4762 - val_accuracy: 0.8734
Epoch 7/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.2295 - accuracy: 0.9422 - val
_loss: 0.4190 - val_accuracy: 0.8955
Epoch 8/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.2252 - accuracy: 0.9373 - val
_loss: 0.4613 - val_accuracy: 0.8761
Epoch 9/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.2205 - accuracy: 0.9397 - val
_loss: 0.3781 - val_accuracy: 0.8941
Epoch 10/30
7352/7352 [==============================] - 3s 378us/step - loss: 0.2105 - accuracy: 0.9408 - val
_loss: 0.3468 - val_accuracy: 0.9050
Epoch 11/30
7352/7352 [==============================] - 3s 386us/step - loss: 0.2099 - accuracy: 0.9422 - val
_loss: 0.4321 - val_accuracy: 0.8778
Epoch 12/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.2127 - accuracy: 0.9414 - val
_loss: 0.4111 - val_accuracy: 0.8935
Epoch 13/30
7352/7352 [==============================] - 3s 378us/step - loss: 0.1965 - accuracy: 0.9444 - val
_loss: 0.3396 - val_accuracy: 0.9192
Epoch 14/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.1907 - accuracy: 0.9463 - val
_loss: 0.3760 - val_accuracy: 0.9070
Epoch 15/30
7352/7352 [==============================] - 3s 382us/step - loss: 0.2043 - accuracy: 0.9419 - val
_loss: 0.3717 - val_accuracy: 0.9067
Epoch 16/30
7352/7352 [==============================] - 3s 386us/step - loss: 0.1932 - accuracy: 0.9452 - val
_loss: 0.4375 - val_accuracy: 0.8683
Epoch 17/30
7352/7352 [==============================] - 3s 382us/step - loss: 0.1888 - accuracy: 0.9467 - val
_loss: 0.3664 - val_accuracy: 0.8989
Epoch 18/30
7352/7352 [==============================] - 3s 388us/step - loss: 0.1923 - accuracy: 0.9482 - val
_loss: 0.3458 - val_accuracy: 0.8894
Epoch 19/30
7352/7352 [==============================] - 3s 379us/step - loss: 0.1880 - accuracy: 0.9453 - val
_loss: 0.3949 - val_accuracy: 0.8914
Epoch 20/30
7352/7352 [==============================] - 3s 392us/step - loss: 0.1810 - accuracy: 0.9487 - val
_loss: 0.3625 - val_accuracy: 0.8955
Epoch 21/30
7352/7352 [==============================] - 3s 398us/step - loss: 0.1884 - accuracy: 0.9452 - val
_loss: 0.3370 - val_accuracy: 0.8941
Epoch 22/30
```

```
7352/7352 [==============================] - 3s 377us/step - loss: 0.1744 - accuracy: 0.9478 - val
_loss: 0.3867 - val_accuracy: 0.8904
Epoch 23/30
7352/7352 [==============================] - 3s 387us/step - loss: 0.1788 - accuracy: 0.9486 - val
_loss: 0.3988 - val_accuracy: 0.8958
Epoch 24/30
7352/7352 [==============================] - 3s 378us/step - loss: 0.1846 - accuracy: 0.9475 - val
_loss: 0.4159 - val_accuracy: 0.8907
Epoch 25/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.1858 - accuracy: 0.9470 - val
_loss: 0.3815 - val_accuracy: 0.9033
Epoch 26/30
7352/7352 [==============================] - 3s 380us/step - loss: 0.1695 - accuracy: 0.9489 - val
_loss: 0.4098 - val_accuracy: 0.8670
Epoch 27/30
7352/7352 [==============================] - 3s 391us/step - loss: 0.1794 - accuracy: 0.9475 - val
_loss: 0.3284 - val_accuracy: 0.8982
Epoch 28/30
7352/7352 [==============================] - 3s 391us/step - loss: 0.1713 - accuracy: 0.9506 - val
_loss: 0.3320 - val_accuracy: 0.8965
Epoch 29/30
7352/7352 [==============================] - 3s 383us/step - loss: 0.1710 - accuracy: 0.9465 - val
_loss: 0.3336 - val_accuracy: 0.9060
Epoch 30/30
7352/7352 [==============================] - 3s 391us/step - loss: 0.1701 - accuracy: 0.9468 - val
_loss: 0.3749 - val_accuracy: 0.8802
```

Out[71]:

```
<keras.callbacks.callbacks.History at 0x2021c149688>
```

1) 88.02% Accuracy

Due to the regularization applied the Validation Loss has decreased.

8.2.2 Adding more Layers

In [3]:

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.python.keras import backend as K
from keras.models import Sequential
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical
from keras.layers import Flatten
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout

# get the features from the file features.txt
features = list()
with open('UCI_HAR_Dataset/features.txt') as f:
    features = [line.split()[1] for line in f.readlines()]
print('No of Features: {}'.format(len(features)))
```

```
No of Features: 561
```

In [4]:

```python
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []
```

```
    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(
            _read_csv(filename).values
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))
```

In [5]:

```
def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return y
```

In [6]:

```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test
```

In [7]:

```
# Importing tensorflow
np.random.seed(42)
tf.random.set_seed(42)
```

In [8]:

```
# Configuring a session
session_conf = tf.compat.v1.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [9]:

```
# Import Keras
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [10]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [11]:

```
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
```

```
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]
```

In [12]:

```
# Loading the train and test data
X_train, X_test, y_train, y_test = load_data()
```

In [13]:

```
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
(7352, 128, 9) (7352,)
(2947, 128, 9) (2947,)
```

In [14]:

```
#https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-
python/#:~:text=A%20one%20hot%20encoding%20is,is%20marked%20with%20a%201.
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
# converting to array
data_y_train = y_train
data_y_train = np.array(data_y_train)

data_y_test = y_test
data_y_test = np.array(data_y_test)

# one hot encode
encoded_y_train = to_categorical(data_y_train)

encoded_y_test = to_categorical(data_y_test)

# invert encoding
inverted_y_train = argmax(encoded_y_train[0])

inverted_y_test = argmax(encoded_y_test[0])

print(data_y_train.shape)
print(data_y_test.shape)

print('********************************')

print(encoded_y_train.shape)
print(encoded_y_test.shape)
```

```
(7352,)
(2947,)
********************************
(7352, 7)
(2947, 7)
```

In [15]:

```
from keras.layers import BatchNormalization
# Importing tensorflow
#np.random.seed(36)
import tensorflow as tf
#tf.set_random_seed(36)
#tf.random.set_seed(36)
```

```python
# Initiliazing the sequential model
model = Sequential()
model.add(Conv1D(32, 3, activation='relu',kernel_initializer = 'he_normal',input_shape=(128, 9)))
#MaxPooling Layer
model.add(MaxPooling1D(2))
# Adding a dropout layer
model.add(Dropout(0.4))
# Adding a Batch Normalization Layer
model.add(BatchNormalization())

model.add(Conv1D(64, 3, activation='relu',kernel_initializer = 'he_normal'))
#model.add(Dropout(0.5))
model.add(MaxPooling1D(2))
model.add(BatchNormalization())

model.add(Conv1D(80, 3, activation='relu',kernel_initializer = 'he_normal'))
model.add(MaxPooling1D(2))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Flatten())

# Adding a dense output layer with sigmoid activation
model.add(Dense(504, activation='relu'))
model.add(BatchNormalization())
#model.add(Dropout(0.5))

model.add(Dense(252, activation= 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(126, activation='relu'))
#model.add(Dropout(0.5))
model.add(BatchNormalization())

model.add(Dense(63, activation='relu'))
model.add(Dropout(0.5))
model.add(BatchNormalization())

model.add(Dense(7, activation='sigmoid'))

model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer='adam')

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 126, 32)           896
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 63, 32)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 63, 32)            0
_________________________________________________________________
batch_normalization_1 (Batch (None, 63, 32)            128
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 61, 64)            6208
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 30, 64)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 30, 64)            256
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 28, 80)            15440
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 14, 80)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 80)            0
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 80)            320
_________________________________________________________________
flatten_1 (Flatten)          (None, 1120)              0
_________________________________________________________________
dense_1 (Dense)              (None, 504)               564984
_________________________________________________________________
batch_normalization_4 (Batch (None, 504)               2016
```

```
_________________________________________________________________
dense_2 (Dense)              (None, 252)               127260
_________________________________________________________________
batch_normalization_5 (Batch (None, 252)               1008
_________________________________________________________________
dropout_3 (Dropout)          (None, 252)               0
_________________________________________________________________
dense_3 (Dense)              (None, 126)               31878
_________________________________________________________________
batch_normalization_6 (Batch (None, 126)               504
_________________________________________________________________
dense_4 (Dense)              (None, 63)                8001
_________________________________________________________________
dropout_4 (Dropout)          (None, 63)                0
_________________________________________________________________
batch_normalization_7 (Batch (None, 63)                252
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 448
=================================================================
Total params: 759,599
Trainable params: 757,357
Non-trainable params: 2,242
_________________________________________________________________
```

In [16]:

```
# Training the model
result = model.fit(X_train,
          encoded_y_train,
          batch_size=10,
          validation_data=(X_test, encoded_y_test),
          epochs=70)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/70
7352/7352 [==============================] - 11s 1ms/step - loss: 1.2301 - accuracy: 0.4943 - val_
loss: 1.0158 - val_accuracy: 0.5114
Epoch 2/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.9202 - accuracy: 0.5974 - val_l
oss: 0.8463 - val_accuracy: 0.5789
Epoch 3/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.7672 - accuracy: 0.6872 - val_l
oss: 0.6689 - val_accuracy: 0.6417
Epoch 4/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.6795 - accuracy: 0.7352 - val_l
oss: 0.5105 - val_accuracy: 0.7703
Epoch 5/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.5388 - accuracy: 0.8092 - val_l
oss: 0.3812 - val_accuracy: 0.8802
Epoch 6/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.5033 - accuracy: 0.8288 - val_l
oss: 0.3838 - val_accuracy: 0.8341
Epoch 7/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.4607 - accuracy: 0.8448 - val_l
oss: 0.3226 - val_accuracy: 0.8700
Epoch 8/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.4327 - accuracy: 0.8535 - val_l
oss: 0.2472 - val_accuracy: 0.9053
Epoch 9/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.3953 - accuracy: 0.8640 - val_l
oss: 0.2765 - val_accuracy: 0.8955
Epoch 10/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.3676 - accuracy: 0.8773 - val_l
oss: 0.2820 - val_accuracy: 0.8992
Epoch 11/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.3613 - accuracy: 0.8807 - val_l
oss: 0.2509 - val_accuracy: 0.8999
Epoch 12/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.3166 - accuracy: 0.8961 - val_l
oss: 0.2424 - val_accuracy: 0.9046
Epoch 13/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.3241 - accuracy: 0.8868 - val_l
oss: 0.2384 - val_accuracy: 0.9108
Epoch 14/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.3015 - accuracy: 0.8889 - val_l
oss: 0.2512 - val_accuracy: 0.9091
```

```
oss: 0.2312 - val_accuracy: 0.9091
Epoch 15/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.3252 - accuracy: 0.8893 - val_l
oss: 0.2275 - val_accuracy: 0.9131
Epoch 16/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2907 - accuracy: 0.9006 - val_l
oss: 0.2411 - val_accuracy: 0.8948
Epoch 17/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2972 - accuracy: 0.8958 - val_l
oss: 0.2279 - val_accuracy: 0.9145
Epoch 18/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2963 - accuracy: 0.8983 - val_l
oss: 0.2191 - val_accuracy: 0.9016
Epoch 19/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2874 - accuracy: 0.9013 - val_l
oss: 0.2536 - val_accuracy: 0.9138
Epoch 20/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2608 - accuracy: 0.9038 - val_l
oss: 0.2240 - val_accuracy: 0.9169
Epoch 21/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2499 - accuracy: 0.9113 - val_l
oss: 0.2310 - val_accuracy: 0.9175
Epoch 22/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2615 - accuracy: 0.9070 - val_l
oss: 0.2086 - val_accuracy: 0.9148
Epoch 23/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2595 - accuracy: 0.9086 - val_l
oss: 0.2752 - val_accuracy: 0.9121
Epoch 24/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2662 - accuracy: 0.9018 - val_l
oss: 0.1971 - val_accuracy: 0.9087
Epoch 25/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2452 - accuracy: 0.9081 - val_l
oss: 0.1981 - val_accuracy: 0.9131
Epoch 26/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2552 - accuracy: 0.9081 - val_l
oss: 0.2207 - val_accuracy: 0.9104
Epoch 27/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2298 - accuracy: 0.9178 - val_l
oss: 0.1987 - val_accuracy: 0.9165
Epoch 28/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2502 - accuracy: 0.9106 - val_l
oss: 0.2192 - val_accuracy: 0.9182
Epoch 29/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2341 - accuracy: 0.9127 - val_l
oss: 0.1877 - val_accuracy: 0.9233
Epoch 30/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2435 - accuracy: 0.9083 - val_l
oss: 0.2142 - val_accuracy: 0.9169
Epoch 31/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2437 - accuracy: 0.9094 - val_l
oss: 0.2201 - val_accuracy: 0.9145
Epoch 32/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2407 - accuracy: 0.9121 - val_l
oss: 0.2075 - val_accuracy: 0.9192
Epoch 33/70
7352/7352 [==============================] - 10s 1ms/step - loss: 0.2353 - accuracy: 0.9081 - val_
loss: 0.2003 - val_accuracy: 0.9186
Epoch 34/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2292 - accuracy: 0.9158 - val_l
oss: 0.2227 - val_accuracy: 0.9226
Epoch 35/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2110 - accuracy: 0.9193 - val_l
oss: 0.2319 - val_accuracy: 0.9074
Epoch 36/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2427 - accuracy: 0.9162 - val_l
oss: 0.2341 - val_accuracy: 0.9131
Epoch 37/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2138 - accuracy: 0.9236 - val_l
oss: 0.2167 - val_accuracy: 0.9101
Epoch 38/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2180 - accuracy: 0.9193 - val_l
oss: 0.2306 - val_accuracy: 0.9077
Epoch 39/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2249 - accuracy: 0.9192 - val_l
oss: 0.2259 - val_accuracy: 0.9152
Epoch 40/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2136 - accuracy: 0.9211 - val_l
```

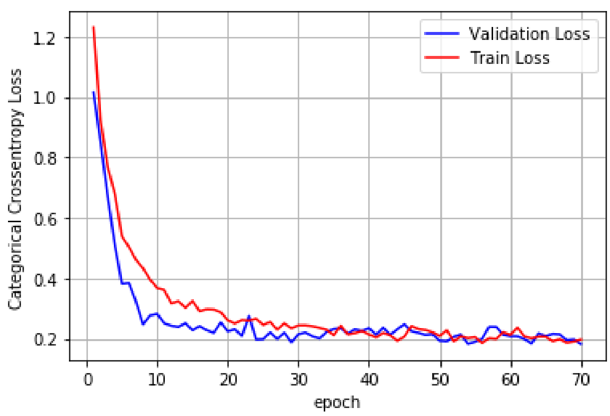
```
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2136 - accuracy: 0.9211 - val_l
oss: 0.2345 - val_accuracy: 0.9196
Epoch 41/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2040 - accuracy: 0.9206 - val_l
oss: 0.2128 - val_accuracy: 0.9145
Epoch 42/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2186 - accuracy: 0.9185 - val_l
oss: 0.2363 - val_accuracy: 0.9165
Epoch 43/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2109 - accuracy: 0.9226 - val_l
oss: 0.2124 - val_accuracy: 0.9203
Epoch 44/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1926 - accuracy: 0.9230 - val_l
oss: 0.2308 - val_accuracy: 0.9182
Epoch 45/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2068 - accuracy: 0.9181 - val_l
oss: 0.2477 - val_accuracy: 0.9118
Epoch 46/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2415 - accuracy: 0.9135 - val_l
oss: 0.2244 - val_accuracy: 0.9111
Epoch 47/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2317 - accuracy: 0.9219 - val_l
oss: 0.2188 - val_accuracy: 0.9277
Epoch 48/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2284 - accuracy: 0.9165 - val_l
oss: 0.2122 - val_accuracy: 0.9237
Epoch 49/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2209 - accuracy: 0.9191 - val_l
oss: 0.2140 - val_accuracy: 0.9226
Epoch 50/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2087 - accuracy: 0.9208 - val_l
oss: 0.1930 - val_accuracy: 0.9277
Epoch 51/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2276 - accuracy: 0.9207 - val_l
oss: 0.1910 - val_accuracy: 0.9253
Epoch 52/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1909 - accuracy: 0.9259 - val_l
oss: 0.2062 - val_accuracy: 0.9162
Epoch 53/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2077 - accuracy: 0.9206 - val_l
oss: 0.2136 - val_accuracy: 0.9264
Epoch 54/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2017 - accuracy: 0.9268 - val_l
oss: 0.1823 - val_accuracy: 0.9213
Epoch 55/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2053 - accuracy: 0.9246 - val_l
oss: 0.1892 - val_accuracy: 0.9257
Epoch 56/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1852 - accuracy: 0.9270 - val_l
oss: 0.1993 - val_accuracy: 0.9182
Epoch 57/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2013 - accuracy: 0.9275 - val_l
oss: 0.2390 - val_accuracy: 0.9220
Epoch 58/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1989 - accuracy: 0.9249 - val_l
oss: 0.2387 - val_accuracy: 0.9131
Epoch 59/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2212 - accuracy: 0.9196 - val_l
oss: 0.2127 - val_accuracy: 0.9192
Epoch 60/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2122 - accuracy: 0.9200 - val_l
oss: 0.2078 - val_accuracy: 0.9226
Epoch 61/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2360 - accuracy: 0.9102 - val_l
oss: 0.2077 - val_accuracy: 0.9199
Epoch 62/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2074 - accuracy: 0.9233 - val_l
oss: 0.1985 - val_accuracy: 0.9172
Epoch 63/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.2010 - accuracy: 0.9257 - val_l
oss: 0.1833 - val_accuracy: 0.9179
Epoch 64/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2066 - accuracy: 0.9202 - val_l
oss: 0.2173 - val_accuracy: 0.9128
Epoch 65/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.2065 - accuracy: 0.9170 - val_l
oss: 0.2080 - val_accuracy: 0.9135
Epoch 66/70
```

```
Epoch 66/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.1911 - accuracy: 0.9255 - val_l
oss: 0.2155 - val_accuracy: 0.9111
Epoch 67/70
7352/7352 [==============================] - 9s 1ms/step - loss: 0.1980 - accuracy: 0.9229 - val_l
oss: 0.2138 - val_accuracy: 0.9216
Epoch 68/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1863 - accuracy: 0.9283 - val_l
oss: 0.1946 - val_accuracy: 0.9145
Epoch 69/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1901 - accuracy: 0.9306 - val_l
oss: 0.1970 - val_accuracy: 0.9281
Epoch 70/70
7352/7352 [==============================] - 8s 1ms/step - loss: 0.1967 - accuracy: 0.9283 - val_l
oss: 0.1821 - val_accuracy: 0.9321
```

In [17]:

```
score = model.evaluate(X_test, encoded_y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,70+1))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, va
lidation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of epochs

vy = result.history['val_loss']
ty = result.history['loss']
ax.plot(x, vy, 'b', label="Validation Loss")
ax.plot(x, ty, 'r', label="Train Loss")
plt.legend()
plt.grid()

fig.canvas.draw()
```

```
Test score: 0.18210736676647027
Test accuracy: 0.9321343898773193
```



In [18]:

```
score = model.evaluate(X_test, encoded_y_test)
print(" \n Loss & Accuracy of test data on Dynamic set :",score )
```

```
2947/2947 [==============================] - 0s 86us/step

 Loss & Accuracy of test data on Dynamic set : [0.18210736676647027, 0.9321343898773193]
```

In [19]:

```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Test Loss","Test Accuracy"]
x.add_row(['Basic LSTM','0.6149','88.90'])
x.add_row(['Adding more layers to LSTM','0.5288','90.13'])
x.add_row(['LSTM+L2 Regularization','0.6374','79.91'])
x.add_row(['LSTM+Hyperas Hyperparameter tuning','0.7085 ','62.84'])
x.add_row(['Basic CNN','0.4972','92.87'])
x.add_row(['CNN+Regularization','0.3749 ','88.02'])
x.add_row(['CNN with More Layers','0.1821','0.9321'])

print(x)
```

```
+------------------------------------+-----------+---------------+
|               Model                | Test Loss | Test Accuracy |
+------------------------------------+-----------+---------------+
|             Basic LSTM             |   0.6149  |     88.90     |
|     Adding more layers to LSTM     |   0.5288  |     90.13     |
|       LSTM+L2 Regularization       |   0.6374  |     79.91     |
| LSTM+Hyperas Hyperparameter tuning |  0.7085   |     62.84     |
|             Basic CNN              |   0.4972  |     92.87     |
|         CNN+Regularization         |  0.3749   |     88.02     |
|        CNN with More Layers        |   0.1821  |     0.9321    |
+------------------------------------+-----------+---------------+
```